ELSEVIER

Science of Computer Programming 28 (1997) 43–90

Science of Computer Programming

Timewise refinement for communicating processes

Steve Schneider *

Royal Holloway, University Of London, Egham, Surrey, TW20 0EX, UK

Received 1 September 1994; revised 1 April 1996
Communicated by R. Bird

Abstract

A theory of timewise refinement is presented. This allows the translation of specifications and proofs of correctness between semantic models, permitting each stage in the verification of a system to take place at the appropriate level of abstraction. The theory is presented within the context of CSP. A denotational characterisation of refinement is given in terms of relations between behaviours at different levels of abstraction. Various properties for the preservation of refinement through parallel composition are discussed. An operational characterisation is also given in terms of timed and untimed tests, and observed to coincide with the denotational characterisation.

Keywords: Concurrency; Real-time systems; Refinement; Timed CSP; Timewise refinement; Verification

1. Introduction and general theory

Verification of time-critical systems requires the application of necessarily complicated and detailed techniques, reflecting the complex nature of such systems and the detailed and precise requirements upon them. Yet it is often the case that a significant proportion of specifications on timed systems will be concerned with logical behaviour rather than timing behaviour, and proposed implementations will often be correct with respect to these parts of the specification by virtue of their functional properties, independently of their timing properties.

Specifications often split naturally into parts that are dependent upon time, and time-independent parts. For example, a timed buffer must meet certain functional constraints: it can output only what has been previously input, and in the same order; it should be deadlock-free, and will eventually become ready to output when non-empty. There might also be requirements on the capacity of the buffer. In addition to these requirements, there might be some desired timing properties, concerning throughput, maximum

* E-mail: steve@dcs.rhbnc.ac.uk.

PII S0167-6423(96)00016-0

rate of input, minimum rate of output, and so on. The techniques required to establish these two kinds of properties will often be quite different, and the timing behaviour of a proposed implementation may be irrelevant in verifying the functional correctness. In such cases, we would hope to perform each part of the verification at the appropriate level of abstraction.

Development of a timed system could also follow this pattern. A system may be developed initially with respect to the time-independent aspects of its specification. A subsequent stage of development may introduce precise timed behaviour into the system to ensure that the timing aspects of the specification are also met. It is important to ensure that the correctness of the first stage is preserved when time is introduced.

This paper proposes a framework in the context of CSP for system development and verification at various levels of abstraction, in such a way that consistency is ensured between results obtained at different levels. We investigate refinement relations between processes in different models of the CSP hierarchy [22]. It is important to identify which properties (such as deadlock-freedom or determinism) can be translated between models, since only for such properties can verifications be mapped up the hierarchy. The more mature and powerful techniques available in the more abstract models (such as model-checking, algebraic techniques, specialised theories for deadlock-freedom, and simply more abstract reasoning) may then be used in conjunction with the more cumbersome and difficult methods required for the more detailed timed aspects of the verification.

This paper investigates two refinement relations in detail, both between an untimed and a timed model of CSP. The first untimed model is concerned only with safety specification. The second is also able to address fairness and (untimed) liveness requirements. The relationships between these two models and the timed infinite model for timed CSP [28, 20] will be presented.

General framework. The various models for CSP all embody a particular approach to denotational semantics. A process is modelled in terms of the observations that may be made of it, which may also be considered as the behaviours it may exhibit. If we have a set $\mathscr{O}$ of all possible observations, then a process is identified with a subset of $\mathscr{O}$. The corresponding semantic model $\mathscr{M}$ consists of those subsets of $\mathscr{O}$ that may be considered to represent some process. A set of healthiness conditions, or axioms, for $\mathscr{M}$ are used to characterise these subsets of $\mathscr{O}$.

A programming language $\mathscr{L}$ is used for describing processes. Each term in $\mathscr{L}$ together with a binding $\rho \in \mathscr{E} = \mathscr{V} \to \mathscr{M}$ of variables to values is associated with an element of $\mathscr{M}$, called its semantics or meaning, by means of a semantic function $\mathscr{F} : \mathscr{L} \to \mathscr{E} \to \mathscr{M}$. This meaning consists of the set of possible behaviours for the term. This function is compositional, in the sense that the process associated with any particular term depends only on the processes associated with its components, and on how these components are composed. A program is a term with no free variables. The process associated with such a program is independent of the environment ρ used to evaluate it.

Following the notation introduced in [13], we give specifications in terms of predicates upon observations. A term P meets a specification $S(o)$ in environment ρ if all of the observations in the semantics of P in ρ meet the corresponding predicate. In this case, we write P **sat**$_\rho$ $S(o)$.

Definition 1.1.

$$P \textbf{ sat}_\rho S(o) \Leftrightarrow \forall o : \mathcal{O} \bullet (o \in \mathcal{F}[\![P]\!]_\rho \Rightarrow S(o))$$

A term P_1 is refined by another term P_2 when every possible behaviour of P_2 is also a possible behaviour of P_1 in any environment. In this case we write $P_1 \sqsubseteq P_2$, and consider P_2 to be more deterministic than P_1, since P_1 can do everything P_2 can, and possibly more. If $P_1 \sqsubseteq P_2$, and P_1 **sat**$_\rho$ $S(o)$, then it follows that P_2 **sat**$_\rho$ $S(o)$; refining a process maintains correctness with respect to specifications. This approach also allows terms P to act as specifications: P_2 meets specification P if it is a refinement of P.

The nature of the semantic model is dependent upon the nature of the observation set $\mathcal{O}$. Observations describe executions of systems at a particular level of abstraction. For example, the use of traces as observations provides only the sequences of events that a system may perform; refusals provide information about contexts in which a system may deadlock; and timed traces also provide information about the times at which events may occur. The use of a particular kind of observation depends on the kind of specification we wish to consider, and the level of abstraction at which we need to consider the system in order to establish correctness.

If we have two different semantic models $\mathcal{M}_A$ and $\mathcal{M}_C$, based upon different sets of observations $\mathcal{O}_A$ and $\mathcal{O}_C$ respectively, then we are able to analyse systems at two different levels of abstraction; and we may ask when a description at the level of $\mathcal{M}_C$ refines a description at the level of $\mathcal{M}_A$. We firstly employ a relation ${}_A\mathcal{R}_C \subseteq \mathcal{O}_A \times \mathcal{O}_C$ to relate observations at the different levels of abstraction. The intention is that if $b_A \; {}_A\mathcal{R}_C \; b_C$ then b_A and b_C are both descriptions, at different levels of abstraction, of the same execution; or alternatively, that b_A is an abstract description of b_C. There is of course no guarantee that the relation ${}_A\mathcal{R}_C$ captures a useful relationship between behaviours; this depends upon the intended application of the theory.

An environment $\rho_A : \mathcal{V} \to \mathcal{M}_A$ is refined by an environment $\rho_C : \mathcal{V} \to \mathcal{M}_C$ if $\rho_A(Y)$ contains every abstract behaviour related to each concrete behaviour in $\rho_C(Y)$. This reflects the idea that the concrete refines the abstract in that it contains fewer behaviours:

Definition 1.2.

$$\rho_A \sqsubseteq_{\mathcal{M}_A \mathcal{M}_C} \rho_C \Leftrightarrow \forall Y : \mathcal{V} \bullet {}_A\mathcal{R}_C^{-1}(\rho_C(Y)) \subseteq \rho_A(Y)$$

The refinement relation may now be given between terms from $\mathcal{M}_A$ and terms from $\mathcal{M}_C$ with respect to the relation ${}_A\mathcal{R}_C$ and a pair of environments:

Definition 1.3.

$$P_A \sqsubseteq_{A\mathscr{R}_C} P_C \Leftrightarrow \forall \rho_A, \rho_C \bullet \rho_A \sqsubseteq_{A\mathscr{R}_C} \rho_C \Rightarrow {}_A\mathscr{R}_C^{-1}\ (\mathscr{F}_C[\![P_C]\!]\rho_C) \subseteq \mathscr{F}_A[\![P_A]\!]\rho_A)$$

The following lemma states that refinement between models is monotonic with respect to refinement within models.

Lemma 1.4.

$$P_{A1} \sqsubseteq_A P_{A2} \sqsubseteq_{A\mathscr{R}_C} Q_{C1} \sqsubseteq_C Q_{C2} \Rightarrow P_{A1} \sqsubseteq_{A\mathscr{R}_C} Q_{C2}$$

A verification of P_A may be translated into a verification of P_C by use of the following inference rule, whose soundness follows from the definitions above:

$$\frac{\begin{array}{l}\rho_A \sqsubseteq_{A\mathscr{R}_C} \rho_C \\ P_A\ \mathbf{sat}_{\rho_A}\ S_A(o_A) \\ P_A \sqsubseteq_{A\mathscr{R}_C} P_C\end{array}}{P_C\ \mathbf{sat}_{\rho_C}\ \forall o_A \bullet (o_A\ {}_A\mathscr{R}_C\ o_C \Rightarrow S_A(o_A))}$$

We may thus consider the specification

$$S_C(o_C) = \forall o_A \bullet (o_A\ {}_A\mathscr{R}_C\ o_C \Rightarrow S_A(o_A))$$

to be the translation of $S_A(o_A)$.

Observe that if $\mathscr{M}_A = \mathscr{M}_C$, and the relation ${}_A\mathscr{R}_C$ is the identity relation, then the refinement relation $\sqsubseteq_{A\mathscr{R}_C}$ is simply refinement under the non-deterministic ordering; and the rule states that if a program meets a specification, then so too does any refinement of it. The refinement relation within a single model $\mathscr{M}_A$ is defined as

$$P_1 \sqsubseteq_A P_2 \Leftrightarrow \forall \rho_A \bullet \mathscr{F}_A[\![P_2]\!]\rho_A \subseteq \mathscr{F}_A[\![P_1]\!]\rho_A$$

The above rule is sound for any refinement relation ${}_A\mathscr{R}_C$, by virtue of the definitions of the operators involved. We shall later take advantage of the dual property of *completeness*, which states that whenever the conclusion to the above rule is true, then a term P_A and environment ρ_A can be found such that the antecedents to the rule are true. In general, this property will not hold, but it may hold for some particular refinement relations ${}_A\mathscr{R}_C$, and in fact we will establish that it holds for the relations presented in this paper.

Definition 1.5. A refinement relation ${}_A\mathscr{R}_C$ is said to be *complete* if whenever the conclusion of the above rule holds for any S_A, then there is some P_A and ρ_A for which the three antecedents hold.

The following lemma provides a necessary and sufficient condition for a refinement relation to be complete, in the case where $\mathscr{F}_A$ is surjective for any ρ_A (as is the case for the untimed semantic models used in this paper).

Lemma 1.6. *The relation* ${}_A\mathscr{R}_C$ *is complete if and only if* ${}_A\mathscr{R}_C^{-1}(T)$ *is an element of* $\mathscr{M}_A$ *whenever* T *is an element of* $\mathscr{M}_C$.

Proof. Assume that ${}_A\mathscr{R}_C^{-1}(T)$ is an element of $\mathscr{M}_A$ whenever T is an element of $\mathscr{M}_C$. Consider the conclusion P_C **sat**$_{\rho_C}$ $\forall o_A \bullet (o_A\ {}_A\mathscr{R}_C\ o_C \Rightarrow S_A(o_A))$. Then it follows that ${}_A\mathscr{R}_C^{-1}(\mathscr{F}_C[\![P_C]\!]\rho_C)$ **sat** $S_A(o_A)$. Define ρ_A by defining it on each Y as $\rho_A(Y) = {}_A\mathscr{R}_C^{-1}(\rho_C(Y))$. Then ρ_A is well-defined since ${}_A\mathscr{R}_C^{-1}(\rho_C(Y))$ is always in $\mathscr{M}_A$. Since ${}_A\mathscr{R}_C^{-1}(\mathscr{F}_C[\![P_C]\!]\rho_C) \in \mathscr{M}_A$ and $\mathscr{F}_A$ is surjective there is some P_A such that $\mathscr{F}_A[\![P_A]\!]\rho_A = {}_A\mathscr{R}_C^{-1}(\mathscr{F}_C[\![P_C]\!]\rho_C)$. Hence all three antecedents of the inference rule hold.

If on the other hand, there is some P_C for which ${}_A\mathscr{R}_C^{-1}(\mathscr{F}_C[\![Q_C]\!]\rho_C) \notin \mathscr{M}_A$, then consider P_A and ρ_A for which $P_A \sqsubseteq_{A\mathscr{R}_C} P_C$. The term P_A will not meet the specification $S_A(o_A)$ defined by $S_A(o_A) = o_A \in {}_A\mathscr{R}_C^{-1}(\mathscr{F}_C[\![P_C]\!]\rho_C)$. This is because ${}_A\mathscr{R}_C^{-1}(\mathscr{F}_C[\![P_C]\!]\rho_C)$ is a proper subset of $\mathscr{F}_A[\![P_A]\!]\rho_A$, and so will contain some behaviour o_A which breaks the specification $S_A(o_A)$; yet the term Q_C in ρ_C meets the translation of that specification:

$$\forall o_A \bullet o_A\ {}_A\mathscr{R}_C\ o_C \Rightarrow S_A(o_A)$$

Hence there is some specification S_A for which the conclusion holds, but for which the second and third antecedents of the rule cannot both hold. □

The definitions given above applied to terms, of which programs are a special case. In the case of programs (but not of general terms) the environment information does not influence the semantics, and environmental considerations may be elided from the definitions and rules below.

The satisfaction relation for programs may be given independently of environmental information:

Definition 1.7.

$$P \textbf{ sat } S(o) \Leftrightarrow \exists \rho \bullet P \textbf{ sat}_\rho S(o)$$

The inference rule given above simplifies to the following in the case of programs:

$$\frac{P_A \textbf{ sat } S_A(o_A) \qquad P_A \sqsubseteq_{A\mathscr{R}_C} P_C}{P_C \textbf{ sat } \forall o_A \bullet (o_A\ {}_A\mathscr{R}_C\ o_C \Rightarrow S_A(o_A))}$$

and is complete in the same circumstances.

Completeness of the refinement relation allows the following rule for programs:

$$\frac{P_C \textbf{ sat } \forall o_A \bullet (o_A\ {}_A\mathscr{R}_C\ o_C \Rightarrow S_A(o_A))}{\exists P_A \bullet P_A \sqsubseteq_{A\mathscr{R}_C} P_C \wedge P_A \textbf{ sat } S_A(o_A)}$$

The discussion so far has all been independent of the details of the programming language. In order to prove that one program refines another using the above theory, it is necessary to calculate the semantics of each program in the appropriate model, and then

check that the refinement relation holds between them. Compositionality often plays a critical role in breaking down verification obligations on large systems to manageable components. We aim to exploit the compositional nature of program semantics, and so we investigate when refinements established between components of abstract and concrete systems mean that the entire abstract system is refined by the entire concrete system.

Our aim is to find relationships concerning the operators of the language $\mathscr{L}$ so that refinement between terms and programs may be established without resorting to explicit calculation of their semantics, by reasoning at the syntactic level.

A syntactic operator $\oplus'$ of $\mathscr{L}$ is a refinement of operator $\oplus$ if $\oplus'$-combinations of refinements of processes refine $\oplus$-combinations of the original processes:

Definition 1.8. An operator $\oplus'$ of the language $\mathscr{L}$ with arity α refines operator $\oplus$ with the same arity, if

$$(\forall i < \alpha \bullet P_i \sqsubseteq_{\mathscr{M}_C} P_i') \Rightarrow \oplus\langle P_i \mid i < \alpha\rangle \sqsubseteq_{\mathscr{M}_C} \oplus'\langle P_i' \mid i < \alpha\rangle$$

In general, this definition applies to two languages $\mathscr{L}$ and $\mathscr{L}'$, but for the purposes of this paper we will work within a single language.

The framework presented above is very well-known. But to go further, we must focus on particular models, languages, and refinement relations. We are interested in conditions for refinement relations to exist between programs (which will vary from relation to relation), and how specifications translate between models.

In this paper we are concerned with mapping results up the hierarchy of untimed and timed models for CSP. We will concentrate on two relations in detail, both from an untimed to a timed model: one from the untimed traces model, which is used for analysis of safety properties; and one from the untimed infinite traces model [24] (which also contains failures and divergences), a more sophisticated untimed model supporting consideration of liveness issues.

2. Communicating sequential processes

2.1. Syntax

The language of communicating sequential processes (CSP) is given as follows:

$$P ::= Loop \mid Stop \mid Skip \mid P; P \mid P \overset{t}{\rhd} P \mid P \,\Box\, P \mid a : A \to P_a \mid \prod_{i \in I} P_i$$
$$\mid P \;{}_A\|_A\; P \mid P \,|||\, P \mid P \setminus A \mid f(P) \mid f^{-1}(P) \mid Y \mid \mu Y \circ P$$

Here the set A is a subset of the universal set of events Σ; I is a subset of the set of indexes $\mathscr{I}$; f is a function $\Sigma \to \Sigma$; Y is drawn from the set of process variables $\mathscr{V}$; and t is drawn from the set of times, the non-negative real numbers. The fixed point operation $\mu Y \circ P$ binds free occurrences of Y in P. The *programs* of the language are

those terms with no free process variables. Observe that the requirement that arguments to an arbitrary internal choice be indexed from $\mathscr{I}$ ensures that the size of the choice is bounded by some cardinal, which is required to ensure that the language is well-defined by this syntax. For more details of the technicalities, see [20].

The constructors given above represent, respectively: the most non-deterministic process; deadlock; successful termination; sequential composition; timeout; external choice; prefix choice; non-deterministic choice; synchronised parallel; interleaving parallel; interface abstraction or hiding; two forms of alphabet renaming; process variable; and recursion. For a more detailed discussion of the language, the reader is referred to [9]. The following abbreviations often prove useful:

$$Wait\, t = Stop \overset{t}{\rhd} Skip$$

$$b \rightarrow P = a : \{b\} \rightarrow P(a) \text{ (where } P(b) \text{ is defined to be } P)$$

$$b \xrightarrow{t} P = b \rightarrow Wait\, t\, ;\, P$$

$$P \| Q = P \;{}_{\Sigma}\|_{\Sigma}\; Q$$

$$P \sqcap Q = \prod_{i \in \{1,2\}} P_i \text{ (where } P_1 \text{ is defined to be } P \text{ and } P_2 \text{ is defined to be } Q)$$

$$P \underset{A}{\|} Q = h_A(\, f_A(P) \;{}_{A \cup 0.\Sigma}\|_{A \cup 1.\Sigma}\; g_A(Q))$$

where

$$f_A(x) = \begin{cases} x & \text{if } x \in A \\ 0.x & \text{otherwise} \end{cases}$$

$$g_A(x) = \begin{cases} x & \text{if } x \in A \\ 1.x & \text{otherwise} \end{cases}$$

$$h_A(x) = \begin{cases} y & \text{if } x = 0.y \\ y & \text{if } x = 1.y \\ x & \text{otherwise} \end{cases}$$

When modelling timed processes, we must take care to ensure that recursive calls are time guarded, so that a minimum delay must elapse between successive recursive calls. This is achieved by ensuring that every instance of the process variable of a recursive term should appear in the right-hand argument of a non-zero timeout. A set of rules for determining when a term is time guarded is detailed in [8].

2.2. Notation

The set Σ is the set of visible events. Variables a, b, c are taken to range over Σ. If $M \subseteq \Sigma$ then $c.M$ is shorthand for the set of events $\{c.m \mid m \in M\} \subseteq \Sigma$. The variable t ranges over $\mathbf{R}_+$, the set of non-negative real numbers. Variable tr ranges over Σ^*, finite sequences of events from Σ; u ranges over Σ^ω, infinite sequences of events from Σ; $X \subseteq \Sigma$ denotes a set of events; s ranges over $(\mathbf{R}_+ \times \Sigma)^* \cup (\mathbf{R}_+ \times \Sigma)^\omega$, the (finite and

infinite) sequences of timed visible events; we use $\aleph \subseteq \mathbf{R}_+ \times \Sigma$ to represent a timed refusal, a set of timed visible events. In fact the timed refusals used within the timed model have a particular structure, as defined later in *IRSET*.

We use the following operations on (untimed and timed) sequences of events: $\#w$ is the length of the sequence w; $w_1 \frown w_2$ denotes the concatenation of w_1 and w_2; $last(w \frown x) = x$ yields the last item in the list. The notation $w_1 \preccurlyeq w_2$ means that w_1 is a (not necessarily contiguous) subsequence of w_2; thc notation $w_1 \leqslant w_2$ means that w_1 is a prefix of w_2; the notation $w_1 \leqslant_i w_2$ means that w_1 is a prefix of w_2 such that $\#w_2 \leqslant \#w_1 + i$.

The following projections are defined on untimed sequences by list and set comprehension:

$$
\begin{aligned}
tr \restriction A &= \langle a \mid a \leftarrow tr, a \in A \rangle \\
tr \setminus A &= \langle a \mid a \leftarrow tr, a \notin A \rangle \\
tr \downarrow c &= \langle x \mid a \leftarrow tr, a = c.x \rangle \\
\sigma(tr) &= \{a \mid tr \restriction \{a\} \neq \langle\rangle\}
\end{aligned}
$$

For timed sequences, we define the beginning and end of a sequence in the following way: $begin(\langle (t,a) \rangle \frown s) = t$, $end(s \frown \langle (t,a) \rangle) = t$, $end(s) = \infty$ when s is infinite, and for convenience $begin(\langle\rangle) = \infty$ and $end(\langle\rangle) = 0$. The following projections on timed sequences are defined by list and set comprehension:

$$
\begin{aligned}
s \lhd t &= \langle (t',a) \mid (t',a) \leftarrow s, t' \leqslant t \rangle \\
s \lhd\!\!\lhd t &= \langle (t',a) \mid (t',a) \leftarrow s, t' < t \rangle \\
s \uparrow t &= \langle (t',a) \mid (t',a) \leftarrow s, t' = t \rangle \\
s \restriction A &= \langle (t',a) \mid (t',a) \leftarrow s, a \in A \rangle \\
s \setminus A &= \langle (t',a) \mid (t',a) \leftarrow s, a \notin A \rangle \\
s - t &= \langle (t'-t,a) \mid (t',a) \leftarrow s, t' \geqslant t \rangle \\
strip(s) &= \langle a \mid (t',a) \leftarrow s \rangle \\
\sigma(s) &= \{a \mid s \restriction \{a\} \neq \langle\rangle\}
\end{aligned}
$$

We also define a number of projections on timed refusal sets:

$$
\begin{aligned}
\aleph \lhd t &= \{(t',a) \mid (t',a) \in \aleph, t' < t\} \\
\aleph \rhd t &= \{(t',a) \mid (t',a) \in \aleph, t' \geqslant t\} \\
\aleph \restriction A &= \{(t',a) \mid (t',a) \in \aleph, a \in A\} \\
\aleph - t &= \{(t'-t,a) \mid (t',a) \in \aleph, t' \geqslant t\} \\
\sigma(\aleph) &= \{a \mid (t',a) \in \aleph\} \\
end(\aleph) &= sup\{t' \mid (t',a) \in \aleph\}
\end{aligned}
$$

We will use $(s, \aleph) - t$ as an abbreviation for $(s - t, \aleph - t)$, and $end(s, \aleph)$ for $max\{end(s), end(\aleph)\}$.

2.3. Semantic models

The hierarchy of models presented in [22] (see Fig. 1) supports reasoning at a number of levels of abstraction, allowing aspects of behaviour dependent upon refusal information, stability information, or timing information to be included as required. In addition to Reed's hierarchy of models, we have the infinite timed model $\mathcal{M}_{TI}$, presented in [28] and [20]; and the untimed infinite traces model $\mathcal{M}_{UI}$ of [24], which is an extension of the failures-divergences model of [3]. In this paper we will focus on the three models which yield the most general results concerning refinement: the untimed traces model $\mathcal{M}_{UT}$, and the two infinite models. These three models are presented in full, together with their corresponding semantic functions, in Appendix A. The semantic functions $\mathcal{F}_{UT}$ and $\mathcal{F}_{UI}$ are surjective in the sense required for completeness in Lemma 1.6; this follows directly from similar results presented in [26].

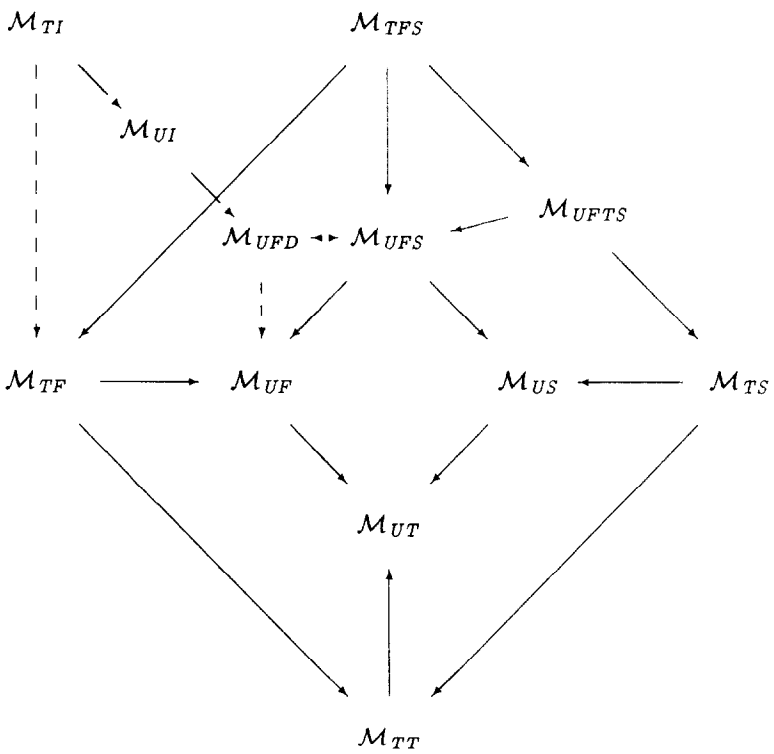


Fig. 1. Reed's hierarchy and additional models.

2.4. The untimed traces model

Observations in the model $\mathcal{M}_{UT}$ are simply finite sequences of events, or *traces*. A trace tr of a system is a record of the events performed during some (partial) execution of the system. Thus, the observation set $\mathcal{O}_{UT}$ is defined to be Σ^*, where Σ is the universal set of events.

The model $\mathcal{M}_{UT}$ is the set of non-empty prefix closed subsets S of $\mathcal{O}_{UT}$.

2.5. The untimed infinite traces model

This model is first described in [24]. In other presentations, processes consist of three components, modelling the three kinds of observation that may be made: a failure set $F \subseteq \Sigma^* \times \mathbf{P}(\Sigma)$; a divergence set $D \subseteq \Sigma^*$; and an infinite traces set $I \subseteq \Sigma^\omega$. A divergence $tr \in D$ is a sequence of events such that after some prefix of tr the system may perform an infinite sequence of internal actions. A failure $(tr, X) \in F$ is an observation of a system if either the sequence of external events tr may be observed during an execution, after which no further internal progress may be made and the process refuses to engage in any event from the set X; or else tr is a divergent trace. An infinite trace u is an infinite sequence of actions such that either the system may perform the whole trace during a single execution, or else some prefix of it is a divergent trace.

For the sake of uniformity within this presentation, we consider a process to consist of a single set S of pairs, where the first component is a label from the set $\{f, d, i\}$, and the second component is a behaviour from the corresponding behaviour set. Thus, S is a subset of

$$\{f\} \times (\Sigma^* \times \mathbf{P}(\Sigma)) \cup \{d\} \times \Sigma^* \cup \{i\} \times \Sigma^\omega$$

2.6. The timed infinite traces model

In this model, the times at which events are performed and refused are recorded. This model assumes that systems are finitely variable: an infinite sequence of internal and external actions may not be performed in a finite time. Thus the only infinite traces that may be observed must take infinitely long to occur. Furthermore, since a change in the set of events made available to the environment is considered to correspond to an internal action, this model needs to consider only those refusal sets which contain finitely many changes in any finite interval.

The set of traces $T\Sigma^\omega_{\leqslant}$ and refusal sets $IRSET$ are adequate for capturing all possible observations of finitely variable systems:

$$T\Sigma^\omega_{\leqslant} = \{s \in (\mathbf{R}^+ \times \Sigma)^* \cup \mathbf{R}^+ \times \Sigma)^\omega \mid \langle (t_1, a_1), (t_2, a_2) \rangle \leqslant s \Rightarrow t_1 \leqslant t_2 \wedge \#s = \infty \Rightarrow end(s) = \infty\}$$

$$RTOK = \{[b, e) \times A \mid 0 \leqslant b < e < \infty \wedge A \subseteq \Sigma\}$$

$$RSET = \left\{ \bigcup R \mid R \subseteq RTOK \wedge R \text{ is finite} \right\}$$

$$IRSET = \left\{ \bigcup R \mid R \subseteq RTOK \wedge \forall t \bullet (\bigcup R) \lhd t \in RSET \right\}$$

Behaviours consist of (trace, refusal) pairs. In contrast to the untimed case, the refusal is observed during the occurrence of the trace, rather than simply afterwards.

For example, the behaviour $(\langle (3,a),(3,d),(8,b)\rangle, [0,20) \times \{c\})$ is a record indicating that the process was observed to refuse event c beginning at time 0, that while it was continuing to do so, it performed event a and then d at time 3, and then event b at time 8. Finally, the observer stopped watching c being refused at time 20.

As usual, a process consists of the set of possible behaviours that may be observed of it. A detailed discussion of the model appears in [28, 20].

Example. Define the program AB as follows:

$$AB = \mu Y \circ (a \xrightarrow{3} Y \overset{5}{\rhd} b \rightarrow Stop)$$

Then $\mathscr{F}_{UT}[\![AB]\!]$ contains both the traces $\langle a\rangle$ and $\langle a,a,b\rangle$, but not trace $\langle b,a\rangle$. The untimed infinite semantics $\mathscr{F}_{UI}[\![AB]\!]$ contains failures $(f, (\langle a,a\rangle, \{a\}))$ and $(f, (\langle a,b\rangle, \{b,c\}))$, but not $(f, \langle a\rangle, \{b\})$ or $(f, \langle b,a\rangle, \{\})$; it contains the infinite trace $(i, \langle a,a,a,\ldots\rangle)$; and it contains no divergences.

The timed behaviours $\mathscr{F}_{TI}[\![AB]\!]$ include $(\langle (2,a),(9,a)\rangle, [0,6) \times \{b\})$: the process may perform event a at time 2, and again at time 9, while refusing to perform b between times 0 and 6. The behaviour $(\langle\rangle, [0,5) \times \{b\} \cup [5,\infty) \times \{a\})$ is also possible: if no external events are performed, then b will be refused for the first 5 units of time, after which the timeout will occur, and a will be refused thereafter. Neither $(\langle (2,a)\rangle, [0,1) \times \{a\})$ nor $(\langle\rangle, [0,10) \times \{b\})$ are possible timed behaviours of $\mathscr{F}_{UI}[\![AB]\!]$.

3. Timewise refinement

3.1. Trace refinement

We consider an untimed trace to be an abstract description of a timed failure if the trace corresponds to the sequence of events in the timed trace. We thus define the refinement relation between untimed traces $\mathscr{C}_{UT}$ and timed failures $\mathscr{C}_{TI}$ as follows:

$$tr \; {}_{UT}\mathscr{R}_{TI} \; (s, \aleph) \Leftrightarrow tr = strip(s)$$

For a timed trace s, the sequence $strip(s)$ is the trace s with the times removed from the events.

Theorem 3.1. *This refinement relation is complete.*

Proof. By Lemma 1.6 it is enough to show that $T = {}_{UT}\mathscr{R}_{TI}^{-1}(U)$ is a well-defined process for any timed process U. But ${}_{UT}\mathscr{R}_{TI}^{-1}(U) = \{strip(s) \mid (s, \aleph) \in U\}$, and this set is clearly non-empty (since U is) and prefix closed (since U is), and hence it is a well-defined process, meeting the definition in Appendix A. □

It turns out that all of the CSP operators preserve this refinement relation:

Theorem 3.2. *Given any CSP operator* $\oplus$, *and two vectors of terms of length* $arity(\oplus)$ $\underline{P}$ *and* $\underline{Q}$ *such that* $\underline{P} \sqsubseteq_{UT\mathscr{R}_{TI}} \underline{Q}$, *then*

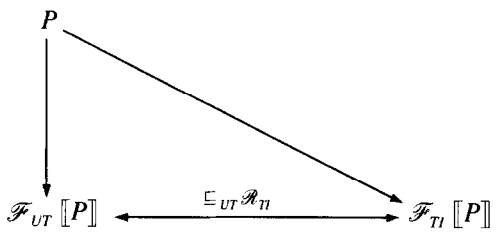
$$\oplus(\underline{P}) \sqsubseteq_{UT\mathscr{R}_{TI}} \oplus(\underline{Q})$$

Proof. By a structural induction over the CSP syntax. The timed and untimed semantics (given in Appendix A) of each CSP operator need to be considered in turn. The case for all operators except recursion follows the same pattern. As an example, we provide the case of the parallel operator. The proof for recursion follows the structure of that given for failure refinement in Lemma 3.5 below, except that in the case here continuity of the CSP operators in $\mathscr{M}_{UT}$ means that a standard induction rather than a transfinite induction is sufficient.

The proof for the parallel operator runs as follows. Assume that $P_i \sqsubseteq_{UT\mathscr{R}_{TI}} Q_i$ for $i = 1, 2$. Then we aim to prove that $P_1 \| P_2 \sqsubseteq_{UT\mathscr{R}_{TI}} Q_1 \| Q_2$.

Consider some $(s, \aleph) \in \mathscr{F}_{TI}[\![Q_1 \| Q_2]\!]\rho_C$. We must prove that $strip(s) \in \mathscr{F}_{UT}[\![P_1 \| P_2]\!]\rho_A$. By the timed semantics of the parallel operator there are $\aleph_i$, $i = 1, 2$ such that $(s, \aleph_i) \in \mathscr{F}_{TI}[\![Q_i]\!]\rho_C$. Since $P_i \sqsubseteq_{UT\mathscr{R}_{TI}} Q_i$ (by the inductive hypothesis) we have $strip(s) \in \mathscr{F}_{UT}[\![P_i]\!]\ \rho_A$ for each i, and hence $strip(s) \in \mathscr{F}_{UT}[\![P_1 \| P_2]\!]\rho_A$ by the untimed trace semantics of the parallel operator. □

Corollary 3.3. *For any program* P, $P \sqsubseteq_{UT\mathscr{R}_{TI}} P$



The payoff from this result is that whenever a trace specification may be verified of a CSP program in the untimed traces model, then it follows immediately that its translation into the timed model will hold for the same program on its more complicated semantics. Also $Skip \sqsubseteq_{UT\mathscr{R}_{TI}} \bigsqcap_{t \in I} Wait\, t$ for any set of times I, so arbitrary delays can be introduced into programs while still preserving refinement, since if $P \sqsubseteq_{UT\mathscr{R}_{TI}} Q$, then it follows that $Skip; P = P \sqsubseteq_{UT\mathscr{R}_{TI}} \bigsqcap_{t \in I} Wait\, t; Q$. Thus, an untimed verification can be carried out and delays inserted subsequently. The translation of a specification $S(tr)$

on traces will be $S'(s,\aleph)$, given by

$$S'(s,\aleph) \Leftrightarrow \forall tr \in \Sigma^* \bullet (tr \ {}_{UT}\mathscr{R}_{TI} \ (s,\aleph) \Rightarrow S(tr))$$

If S is admissible (i.e. $(\forall tr < u \bullet S(tr)) \Rightarrow S(u)$ for every infinite trace u) then the translation is equivalent on processes to the specification $S''(s,\aleph) = S(strip(s))$. Thus admissible specifications may be translated to timed specifications by a simple substitution of the free variable. Since most safety specifications are admissible, this does not amount to a practical limitation.

As an example, consider the safety requirement that no event should be performed after a b. This is given by

$$S(tr) = \forall tr_0, tr_1 \bullet (tr = tr_0 \frown \langle b \rangle \frown tr_1) \Rightarrow tr_1 = \langle\rangle$$

A verification in the traces model that program AB satisfies this specification would be quite straightforward. We may translate this verification to the timed model, and conclude that AB **sat** $S(strip(s))$ in that model. This may then be used in a timed verification. For example, consider the timed specification that event a should never be performed within 8 time units of any b:

$$\langle (t,b) \rangle \preccurlyeq s \ \Rightarrow \ a \notin \sigma(s \uparrow (t-8, t+8))$$

This specification reads as follows: if (t,b) is recorded in the trace s, then a does not appear in the set of events recorded in s during the interval $(t-8, t+8)$. Then the untimed specification tells us that a cannot occur after b, i.e. in the interval $(t, t+8)$, so the only cases to consider in the timed model are a occurring before b, or at the same time, i.e. the interval $(t-8, t]$. For this case, a timed analysis on AB is required.

In general (even when $S(tr)$ is inadmissible), $S(tr)$ is translated to $(\#s < \infty \Rightarrow S(strip(s))$.

3.2. Failures refinement

We think of a process refusing a particular set, in the untimed sense, if it eventually reaches a state after which no event from that set is possible. In the timed world, this corresponds to the information that there is some time after which the set may be continuously refused. Thus for a timed behaviour $(s,\aleph)$ with finite timed trace s, an abstract view of this behaviour would be an untimed version $strip(s)$ of the trace, and for any set X, if there is some t for which $[t,\infty) \times X$ is contained in $\aleph$, then $\aleph$ is evidence that X may eventually be refused forever.

$$(f,(tr,X))_{UI}\mathscr{R}_{TI}(s,\aleph) \ \Leftrightarrow \ tr = strip(s) \wedge \exists t \bullet [t,\infty) \times X \subseteq \aleph$$

Relating timed infinite traces to untimed ones, we obtain the following refinement relation between $\mathscr{C}_{UI}$ and $\mathscr{C}_{TI}$:

$$(i,u)_{UI}\mathscr{R}_{TI}(s,\aleph) \ \Leftrightarrow \ u = strip(s)$$

Observe that there is no timed version of divergence in this model.

Theorem 3.4. *This refinement relation is complete.*

Proof. We need to show that $T = {}_{UI}\mathscr{R}_{TI}^{-1}(U)$ is a well-defined process for any timed process U (i.e. T meets axioms 1–8 given in Appendix A).

Axiom 2 for $U \in \mathscr{M}_{TI}$ yields axioms 1, 2 and 6 for T; axioms 4, 5 and 7 are trivial for T since T has no behaviours of the form (d, tr).

Axiom 3 for U yields axiom 3 for T: if $(f,(tr,X)) \in T \wedge \forall a \in Y \bullet (f,(tr^\frown\langle a\rangle,\{\})) \notin T$ then $(s,[t_0,\infty) \times X) \in U$ for some $t_0 > end(s)$, and also for any $a \in Y$ and any t we have $(s^\frown\langle(t,a)\rangle,\{\}) \notin U$ (since their untimed counterparts are not in T). Now by axiom 3 for U there is some $\aleph' \supseteq \aleph$ which contains all unperformable events. Assume for a contradiction that $(t,a) \notin \aleph'$ for some $t \geqslant t_0$ and $a \in Y$. Then $((s \lhd t)^\frown\langle(t,a)\rangle, \aleph' \lhd t) \in U$, and so $(s^\frown\langle(t,a)\rangle,\{\}) \in U$ by axiom 2 (since $s = s \lhd t$), yielding the contradiction. Hence $(t,a) \in \aleph'$ whenever $t \geqslant t', a \in Y$, yielding $[t_0,\infty) \times Y) \subseteq \aleph'$, and also $[t_0,\infty) \times (X \cup Y) \subseteq \aleph'$, and so $(s,[t_0,\infty) \times (X \cup Y)) \in U$, and hence $(f,(tr,X \cup Y)) \in S$, establishing that axiom 3 for S holds.

It remains only to establish axiom 8 for S. Consider a behaviour $(f,(s,\{\})) \in T$. Then there must be some timed trace s_0 such that $(s_0,\{\}) \in U$ and $strip(s_0) = s$. Now by axiom 4 for $\mathscr{M}_{TI}$ there must be some process $U' \in \mathscr{CL}$ such that $(s_0,\{\}) \in U'$ and $U' \subseteq U'$.

Let $c : \mathbf{P}(\mathbf{R}) \to \mathbf{R}$ be a choice function. Then define:

$$
\begin{aligned}
V_0 &= \{s_0\} \\
V_{n+1} &= \{s^\frown\langle(t_a,a)\rangle \mid s \in V_n \wedge a \in \Sigma \\
&\qquad \wedge \{t \mid (s^\frown\langle(t,a)\rangle,\{\}) \in U'\} \neq \{\} \\
&\qquad \wedge t_a = c(\{t \mid (s^\frown\langle(t,a)\rangle,\{\}) \in U'\}
\end{aligned}
$$

$$
V = \overline{\bigcup_{i \in \mathbf{N}} T_i}
$$

Since U' is finitely variable and closed, any infinite trace in V comes from a legitimate infinite trace in U'. And since U' meets axiom 3 for $\mathscr{M}_{TI}$, the set of all events that do not extend a given finite trace in V must be refusable for all time after the corresponding timed trace in U', since there is no time after that timed trace at which any of those events is possible. It follows that V is a set of traces that establishes that axiom 8 holds for T. □

A study of the CSP operators reveals the following:

Theorem 3.5. *Given any CSP operator $\oplus$ except parallel composition, and two vectors of terms of length arity($\oplus$) $\underline{P}$ and $\underline{Q}$ such that $\underline{P} \sqsubseteq_{UT\mathscr{R}_{TI}} \underline{Q}$, then*

$$\oplus(\underline{P}) \sqsubseteq_{UT\mathscr{R}_{TI}} \oplus(\underline{Q})$$

Proof. The structure of the cases of this proof is entirely similar to that of Lemma 3.2. All operators follow the same pattern, with the exception of the proof of recursion, which we now give.

Assume that $P \sqsubseteq_{UT\mathscr{R}_{TI}} Q$. Now consider a particular pair of environments $\rho_A \sqsubseteq_{UT\mathscr{R}_{TI}} \rho_C$. Let $U = (\mathscr{F}_{TI}[\![\mu Y \circ Q]\!]\rho_C)$ and $T = {}_{UT}\mathscr{R}_{TI}^{-1}(U)$. We must prove $T \subseteq \mathscr{F}_{UI}[\![\mu Y \circ P]\!]\rho_A$. Observe that $T \in \mathscr{M}_{UI}$ by the proof of Theorem 3.4.

We use the fact that $\mathscr{F}_{UI}[\![\mu Y \circ P]\!]\rho_A = F^{\alpha}(\perp)$ for some ordinal α, where $\perp = \mathscr{F}_{UI}[\![Loop]\!]\rho_A$, and

$$F(S) = \mathscr{F}_{UI}[\![P]\!](\rho[S/Y])$$

Hence it is enough to prove by transfinite induction that $T \subseteq F^{\alpha}(\perp)$ for all ordinals α.
Base case: trivial, since any element of $\mathscr{M}_{UI}$ is contained in $\perp$.
Successor case: if $T \subseteq F^{\alpha}(\perp)$, then

$$\begin{aligned}
T &= {}_{UT}\mathscr{R}_{TI}^{-1}(U) \\
&\quad U \text{ is a fixed point of } \lambda S \bullet \mathscr{F}_{TI}[\![Q]\!]\rho_C[S/Y] \\
&= {}_{UT}\mathscr{R}_{TI}^{-1}(\mathscr{F}_{TI}[\![Q]\!](\rho_C[U/Y])) \\
&\quad \text{structural inductive hypothesis: } P \sqsubseteq_{UT\mathscr{R}_{TI}} Q,\ \rho_A[T/Y] \sqsubseteq_{UT\mathscr{R}_{TI}} \rho_C[U/Y] \\
&\subseteq \mathscr{F}_{UI}[\![P]\!](\rho_A[T/Y]) \\
&\quad \text{transfinite inductive hypothesis, and monotonicity of all CSP operators} \\
&\subseteq \mathscr{F}_{UI}[\![P]\!](\rho_A[F^{\alpha}(\perp)/Y] \\
&\quad \text{definition} \\
&= F^{\alpha+1}(\perp)
\end{aligned}$$

as required.
Limit case: if γ is a limit ordinal, then the inductive hypothesis states that $T \subseteq F^{\alpha}(\perp)$ for all $\alpha < \gamma$. Then $F^{\gamma}(\perp) = \bigsqcup_{\alpha<\gamma} F^{\alpha}(\perp)$. Since T is an upper bound for each element of the chain, and $T \in \mathscr{M}_{UI}$, then the least upper bound of the chain $F^{\gamma}(\perp)$ is defined, and $T \subseteq F^{\gamma}(\perp)$, as required.
This establishes the case for recursion. □

We again obtain that $Skip \sqsubseteq_{UT\mathscr{R}_{TI}} \bigsqcap_{t\in I} Wait\, t$ for any set of times I, so arbitrary delays can be introduced into programs while still preserving timewise failures refinement.

Unfortunately, parallel composition does not preserve refinement in general. One example where it fails is in the case of two programs Q_1 and Q_2, illustrated in Fig. 2. They are always willing to perform an event at some time in the future, by offering it periodically (so neither will eventually always refuse it), but they are unable to find any time on which they can synchronise, so their combination is able to refuse the offer forever:

$$\begin{aligned}
Q_1 &= \mu Y \circ (a \rightarrow Stop) \overset{1}{\triangleright} Wait\, 3; Y \\
Q_2 &= Wait\, 2; Q_1
\end{aligned}$$

$$Q_1 = \mu Y \circ (a \longrightarrow Stop) \overset{1}{\triangleright} Wait\,3 ; Y$$
$$Q_2 = Wait\,2 ; Q_1$$

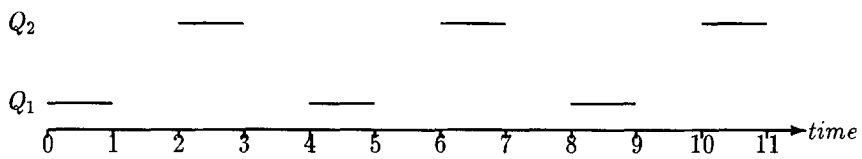


Fig. 2. Alternating offers.

Each of Q_1 and Q_2 are refinements of $P = a \rightarrow Stop$, but $Q_1 \| Q_2$ is not a refinement of $P \| P$, since it may refuse a forever, as Q_1 and Q_2 can never synchronise on a; yet $P \| P$ is unable to refuse a.

However, we will later obtain the following theorem (Theorem 4.9).

Theorem 3.6. *Every CSP program P has that $P \sqsubseteq_{UI\mathscr{R}_{TI}} P$.*

Observe that this theorem holds for $Q_1 \| Q_2$, since the untimed semantics reveals a divergence in each of Q_1 and Q_2: a possible infinite sequence of internal events (repeatedly timing out and passing round the loop without the performance of event a). In the failures model, the untimed semantics is that of *Loop* which is refined by any process. In fact, any program which could repeatedly make and retract offers in the timed model would be divergent in the failures model (though as in the case of Q_1 and Q_2, it could also refine a non-divergent program).

3.1.1. Parallel composition and refinement

The importance of the parallel operator moves us to investigate conditions under which it does preserve refinement. The results presented below concern programs, but they generalise to terms in particular environments, in the obvious way.

3.1.2. Non-retraction

The example above illustrates one of the ways in which refinement may be lost by parallel composition: the periodic withdrawal of offers. One way to ensure synchronisation is to maintain offers until they are accepted.

A process which does not withdraw offers (though it may make new ones) until it next performs a visible event is termed *non-retracting*. This is similar to (though slightly weaker than) the notion of *nonpre-emptive* given in [5], although that definition is given in operational terms.

Definition 3.7. A program Q is *non-retracting* if

$$(s, \aleph) \in \mathscr{F}_{TI}[\![Q]\!] \Rightarrow (s, \aleph \cup \{(t, a) \mid \exists t' \bullet (t', a) \in \aleph \wedge end(s) \leqslant t < t'\}) \in \mathscr{F}_{TI}[\![Q]\!]$$

If an event may be refused at a time t', then it must be possible that it was continuously refused since the occurrence of the last visible event, at time $end(s)$. Thus once an event is guaranteed to have been offered, it must be continually offered thereafter.

As expected, we obtain that parallel composition preserves refinement for non-retracting programs:

Lemma 3.8. *If* $P_1 \sqsubseteq_{UI\mathscr{R}_{TI}} Q_1$ *and* $P_2 \sqsubseteq_{UI\mathscr{R}_{TI}} Q_2$, *and* Q_1 *and* Q_2 *are both non-retracting, then* $(P_1 \| P_2) \sqsubseteq_{UI\mathscr{R}_{TI}} (Q_1 \| Q_2)$

Proof. This is a special case of Lemma 3.10 below, with $A_1 = A_2 = \Sigma$, and the fact that non-retraction is stronger than eventual non-retraction on Σ. □

This idea of non-retraction may be generalised, so that it is concerned only with particular events rather than all events, and with the fact that a time after which offers should be maintained is reached eventually rather than immediately.

Definition 3.9. A program Q is *eventually non-retracting on* A if for any trace s there is some time $t(s)$ such that

$$(s, \aleph) \in \mathscr{F}_{TI}[\![Q]\!] \Rightarrow (s, \aleph \cup \{(t, a) \mid a \in A \wedge \exists t' \bullet (t', a) \in \aleph \wedge t(s) \leqslant t < t'\}) \in \mathscr{F}_{TI}[\![Q]\!]$$

Observe that if S is eventually non-retracting on A, and $B \subseteq A$, then it is also eventually non-retracting on B.

This form of eventual non-retraction allows a period of unstable behaviour before settling down. This permits some timeout behaviour disallowed by a non-retraction requirement. For example, $a \rightarrow P \overset{3}{\rhd} b \rightarrow Q$ is eventually non-retracting (if P and Q are) although the offer of a will be retracted at time 3.

Lemma 3.8 may be generalised to interface parallel, by considering programs that are non-retracting on their common interface.

Lemma 3.10. *If* $P_1 \sqsubseteq_{UI\mathscr{R}_{TI}} Q_1$ *and* $P_2 \sqsubseteq_{UI\mathscr{R}_{TI}} Q_2$, *and* Q_1 *and* Q_2 *are both eventually non-retracting on* $A_1 \cap A_2$, *then*

$$(P_1 \;{}_{A_1}\|_{A_2}\; P_2) \sqsubseteq_{UI\mathscr{R}_{TI}} (Q_1 \;{}_{A_1}\|_{A_2}\; Q_2)$$

Proof. If $(i, u)_{UI}\mathscr{R}_{TI}(s, \aleph)$ and $(s, \aleph) \in \mathscr{F}_{TI}[\![Q_1 \;{}_{A_1}\|_{A_2}\; Q_2]\!]$, then it follows immediately that $(i, u \upharpoonright A_1) \in \mathscr{F}_{UT}[\![P_1]\!] \wedge (i, u \upharpoonright A_2) \in \mathscr{F}_{UT}[\![P_2]\!]$, and hence that $(i, u) \in \mathscr{F}_{UI}[\![P_1 \;{}_{A_1}\|_{A_2}\; P_2]\!]$.

Consider $(f, (tr, X))$ related via ${}_{UI}\mathscr{R}_{TI}$ to $(s, \aleph) \in \mathscr{F}_{TI}[\![Q_1 \;{}_{A_1}\|_{A_2}\; Q_2]\!]$, with $(f, (tr, X))$ ${}_{UI}\mathscr{R}_{TI}(s, \aleph)$. Then $tr = strip(s)$, and $\exists t \bullet [t, \infty) \times X \subseteq \aleph$. Now by the semantics of the parallel operator, there are $\aleph_1$ and $\aleph_2$ such that $(s \upharpoonright A_1, \aleph_1) \in \mathscr{F}_{TI}[\![Q_1]\!]$, $(s \upharpoonright A_2, \aleph_2) \in \mathscr{F}_{TI}[\![Q_2]\!]$, and $\aleph \upharpoonright (A_1 \cup A_2) = (\aleph_1 \upharpoonright A_1) \cup (\aleph_2 \upharpoonright A_2)$. Define

$$X_1 = \{a \in A_1 \mid \forall t \bullet a \in \sigma(\aleph_1 \rhd t)\}$$

$$X_2 = \{a \in A_2 \mid \forall t \bullet a \in \sigma(\aleph_2 \rhd t)\}$$

Then $X_1 \cup X_2 = X \upharpoonright (A_1 \cup A_2)$. Furthermore, by the eventual non-retraction of Q_1 and Q_2, it follows that there are t_1 and t_2 such that

$$(s \upharpoonright A_1, \aleph_1 \cup [t_1, \infty) \times X_1)) \in \mathscr{F}_{TI}[\![Q_1]\!]$$
$$(s \upharpoonright A_2, \aleph_2 \cup [t_2, \infty) \times X_2)) \in \mathscr{F}_{TI}[\![Q_2]\!]$$

Since each P_i is refined by the corresponding Q_i, it follows that

$$(f, (strip(s) \upharpoonright A_1, X_1)) \in \mathscr{F}_{UI}[\![P_1]\!]$$
$$(f, (strip(s) \upharpoonright A_2, X_2)) \in \mathscr{F}_{UI}[\![P_2]\!]$$

and so $(f, (strip(s), X_1 \cup X_2)) \in \mathscr{F}_{UI}[\![P_1 \ {}_{A_1}\|_{A_2} \ P_2]\!]$ by the semantics of the parallel operator. Hence, the parallel operator preserves refinement for eventually non-retracting programs. □

However, if only one of the programs is non-retracting, then the refinement need not be preserved through a parallel combination. For example, consider the following programs, illustrated in Fig. 3 (where *succ* is an alphabet renaming which maps n to $n + 1$).

$$Q_1 = Wait\,2; \mu Y \circ (0 \rightarrow Stop \ \Box \ Wait\,1; succ(Y))$$
$$Q_2 = \mu Y \circ (n : \mathbf{N} \rightarrow Stop) \overset{1}{\triangleright} succ(Y)$$

The process described by Q_1 makes natural numbers available, one at a time. Q_1 is non-retracting, and it is also a refinement of $P_1 = n : \mathbf{N} \rightarrow Stop$; on the empty trace, nothing may be refused forever. The process described by Q_2 begins with all natural numbers available, and retracts them one at a time. Q_2 is a refinement of

$$P_2 = \prod_{F \subseteq^{fin} \mathbf{N}} n : (\mathbf{N} \setminus F) \rightarrow Stop$$

$$Q_1 = Wait\,2\,;\,\mu\,Y \circ (0 \longrightarrow Stop \,\Box\, Wait\,1\,;\,succ(Y))$$
$$Q_2 = \mu\,Y \circ (n : \mathbf{N} \longrightarrow Stop) \overset{1}{\triangleright} succ(Y)$$

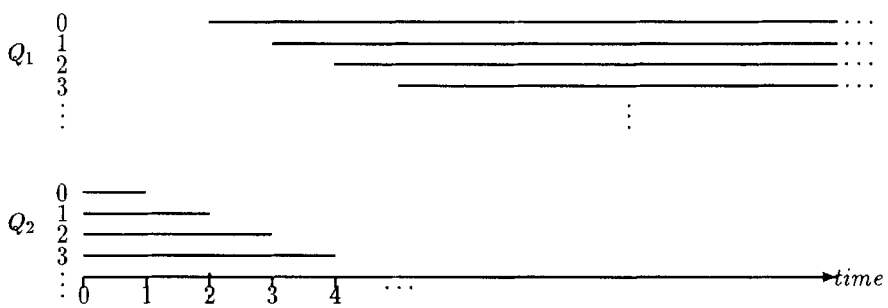


Fig. 3. Non-synchronising offers.

for which all events are possible, and any finite set of events may eventually be refused forever. The parallel combination $Q_1 \| Q_2$ is semantically equivalent to *Stop*, since there is no event that Q_1 and Q_2 may cooperate on: Q_1 is prepared to perform event m from time $m+2$ onwards, but Q_2 is not prepared to perform it beyond time $m+1$. On the other hand, $P_1 \| P_2$ is equivalent to P_2, which cannot deadlock before any events have been performed. Hence, $Q_1 \| Q_2$ is not a refinement of $P_1 \| P_2$, even though Q_1 is non-retracting.

3.1.3. Promptness

The above example highlights other ways in which parallel combination can fail to preserve refinement. If Q_1 had made all of its offers by some time t, then the counterexample would not work, since the non-retraction of Q_1 would ensure that all of the offers, made by time t, must remain on offer until acceptance occurs.

We define a program to be t-prompt if it must make its offers within t time units of the end of the trace: if a set may be refused for time t, then it may be refused forever.

Definition 3.11. A program Q is *t-prompt* if

$$(s, \aleph) \in \mathcal{F}_{TI}[\![Q]\!] \wedge [t', t'+t) \times A \subseteq \aleph \wedge t' \geqslant end(s) \Rightarrow (s, \aleph \cup [t', \infty) \times A) \in \mathcal{F}_{TI}[\![Q]\!]$$

A program is *prompt* if it is t-prompt for some t.

We then obtain the following alternative result, which places no constraints upon Q_2:

Lemma 3.12. *If* $P_1 \sqsubseteq_{U,\mathcal{R}_{TI}} Q_1$ *and* $P_2 \sqsubseteq_{U,\mathcal{R}_{TI}} Q_2$, Q_1 *is non-retracting and t-prompt for some t, then* $(P_1 \| P_2) \sqsubseteq_{U,\mathcal{R}_{TI}} (Q_1 \| Q_2)$.

Proof. This is a special case of Lemma 3.14 below. □

Promptness may be generalised to apply only to a particular set of events A.

Definition 3.13. A program Q is *t-prompt on A* if

$$(s, \aleph) \in \mathcal{F}_{TI}[\![Q]\!] \wedge B \subseteq A \wedge [t', t'+t) \times B \subseteq \aleph \wedge t' \geqslant end(s)$$
$$\Rightarrow (s, \aleph \cup [t', \infty) \times B) \in \mathcal{F}_{TI}[\![Q]\!]$$

Then the condition for parallelism to preserve refinement may be correspondingly generalised: if the interface between two programs may be split into two parts, and each program is prompt and non-retracting on a different part, then refinement will be preserved by parallel composition.

Lemma 3.14. *If* $P_1 \sqsubseteq_{U,\mathcal{R}_{TI}} Q_1$, $P_2 \sqsubseteq_{U,\mathcal{R}_{TI}} Q_2$, $B_1 \cup B_2 = A_1 \cap A_2$, Q_1 *is eventually non-retracting on* B_1 *and prompt on* B_1, *and* Q_2 *is eventually non-retracting on* B_2 *and prompt on* B_2, *then*

$$(P_1 \|_{A_1} \|_{A_2} P_2) \sqsubseteq_{U,\mathcal{R}_{TI}} (Q_1 \|_{A_1} \|_{A_2} Q_2)$$

Proof. It is clear that infinite traces of the timed parallel combination will appear in untimed form in the untimed combination. So we need only show that

$$(f,(tr,X))_{UI}\mathscr{R}_{TI}(s,\aleph) \wedge (s,\aleph) \in \mathscr{F}_{TI}[\![Q_1 \ {}_{A_1}\|_{A_2} \ Q_2]\!]$$
$$\Rightarrow (f,(tr,X)) \in \mathscr{F}_{UI}[\![P_1 \ {}_{A_1}\|_{A_2} \ P_2]\!]$$

Assume the antecedent, and consider $(s,\aleph) \in \mathscr{F}_{TI}[\![Q_1 \ {}_{A_1}\|_{A_2} \ Q_2]\!]$, built from behaviours $(s \upharpoonright A_1, \aleph_1) \in \mathscr{F}_{TI}[\![Q_1]\!]$ and $(s \upharpoonright A_2, \aleph_2) \in \mathscr{F}_{TI}[\![Q_2]\!]$, where we have $(\aleph_1 \upharpoonright A_1) \cup (\aleph_2 \upharpoonright A_2) = \aleph \upharpoonright (A_1 \cup A_2)$. We have by the relation ${}_{UI}\mathscr{R}_{TI}$ that $tr = strip(s)$, and also there is some t such that $t \geqslant end(s) + \max\{t(s \upharpoonright A_1, Q_1), t(s \upharpoonright A_2, Q_2)\}$ for some t, such that $[t,\infty) \times X \subseteq \aleph$. Now Q_1 is t_1-prompt for some t_1, and Q_2 is t_2-prompt for some t_2. Define

$$X_1 = \sigma((\aleph_1 \upharpoonright (X \cap A_1 \cap B_1)) \rhd (t_1 + t))$$
$$Y_1 = \{a \in X \cap A_1 \cap B_2 \mid [t_2 + t, \infty) \times \{a\} \subseteq \aleph_1\}$$
$$X_2 = \sigma((\aleph_2 \upharpoonright (X \cap A_2 \cap B_2)) \rhd (t_2 + t))$$
$$Y_2 = \{a \in X \cap A_2 \cap B_1 \mid [t_1 + t, \infty) \times \{a\} \subseteq \aleph_2\}$$

Then $X_1 \cup Y_1 \cup X_2 \cup Y_2 = X \cap (A_1 \cup A_2)$. Now since Q_1 is eventually non-retracting on B_1, it follows (also using subset closure of refusals for Q_1) that

$$(s \upharpoonright A_1, \aleph_1 \cup [t, t + t_1) \times X_1) \in \mathscr{F}_{TI}[\![Q_1]\!]$$

And hence it follows by promptness that

$$(s \upharpoonright A_1, \aleph_1 \cup [t, \infty) \times X_1) \in \mathscr{F}_{TI}[\![Q_1]\!]$$

Also, observe that $[t_2 + t, \infty) \times Y_1 \subseteq \aleph_1$. Since Q_1 is a refinement of P_1 it follows that $(f,(tr \upharpoonright A_1, X_1 \cup Y_1)) \in \mathscr{F}_{UI}[\![P_1]\!]$. Similarly we obtain that $(f,(tr \upharpoonright A_2, X_2 \cup Y_2)) \in \mathscr{F}_{UI}[\![P_2]\!]$. Thus from the semantics of parallel, we obtain that $(f,(tr,X)) \in \mathscr{F}_{UI}[\![P_1 \ {}_{A_1}\|_{A_2} \ P_2]\!]$, yielding the result. □

This result is particularly useful, as it applies immediately to systems such as those described in terms of input/output automata [18] where input is always possible, and so components are always non-retracting and prompt on input. In these systems, parallel composition connects outputs from one program to corresponding inputs of the other. Thus, the interface in a parallel composition may be partitioned into those events input by one component, and those input by the other. The two programs will be prompt and non-retracting on these two sets respectively.

This result may also be applied to CSP descriptions of occam programs, since in such programs output guards are not permitted. Once a program is prepared to perform an output, it remains ready to perform it until it occurs. Consequently, program are always non-retracting on output, so parallel composition will preserve refinement for prompt components.

3.1.4. Compactness

The final condition we will present here concerns the nature of the untimed programs P_1 and P_2. A process is compact if its refusals are determined (in a particular way) by the finite refusal sets. If the untimed processes are compact, then it turns out that only one of the timed processes need be non-retracting for refinement to be preserved by the parallel operator. In the example above, P_2 fails this condition; any finite subset of $\mathbf{N}$ may be refused, but infinite subsets may not.

Definition 3.15. A program P is *compact* if for any $tr \in \Sigma^*, Y \subseteq \Sigma$ we have

$$(\forall X \subseteq^{fin} Y \bullet (f,(tr,X)) \in \mathscr{F}_{UI}[\![P]\!]) \Rightarrow (f,(tr,Y)) \in \mathscr{F}_{UI}[\![P]\!]$$

Lemma 3.16. *If* $P_1 \sqsubseteq_{UI\mathscr{R}_{TI}} Q_1$, $P_2 \sqsubseteq_{UI\mathscr{R}_{TI}} Q_2$, Q_1 *is eventually non-retracting on* $A_1 \cap A_2$, *and* P_1, P_2 *are compact, then*

$$(P_1 \ {}_{A_1}\|_{A_2} P_2) \sqsubseteq_{UI\mathscr{R}_{TI}} (Q_1 \ {}_{A_1}\|_{A_2} Q_2)$$

Proof. It is clear that infinite traces of the timed process will appear in the untimed process. So we need only show that

$$\begin{aligned}&(f,(tr,X))_{UI}\mathscr{R}_{TI}(s,\aleph) \wedge (s,\aleph) \in \mathscr{F}_{TI}[\![Q_1 \ {}_{A_1}\|_{A_2} Q_2]\!] \\ &\quad \Rightarrow (f,(tr,X)) \in \mathscr{F}_{UI}[\![P_1 \ {}_{A_1}\|_{A_2} P_2]\!]\end{aligned}$$

Consider $(s,\aleph) \in \mathscr{F}_{TI}[\![Q_1 \ {}_{A_1}\|_{A_2} Q_2]\!]$, built from behaviours $(s \upharpoonright A_1, \aleph_1) \in \mathscr{F}_{TI}[\![Q_1]\!]$ and $(s \upharpoonright A_2, \aleph_2) \in \mathscr{F}_{TI}[\![Q_2]\!]$, where $\aleph_1 \upharpoonright A_1 \cup \aleph_2 \upharpoonright A_2 = \aleph \upharpoonright (A_1 \cup A_2)$. We have by the relation ${}_{UI}\mathscr{R}_{TI}$ that $tr = strip(s)$, and also that there is some $t \geqslant end(s) + t(s \upharpoonright A_1, Q_1)$ such that $[t,\infty) \times X \subseteq \aleph$. Define

$$\begin{aligned}X_1 &= \{a \in X \cap A_1 \mid \forall t' \bullet a \in \sigma(\aleph_1 \rhd t')\} \\ X_2 &= \{a \in X \cap A_2 \mid \exists t' \bullet [t',\infty) \times \{a\} \subseteq \aleph_2\}\end{aligned}$$

Since $\aleph_1 \upharpoonright A_1 \cup \aleph_2 \upharpoonright A_2 = \aleph \upharpoonright (A_1 \cup A_2)$, we have that $X_1 \cup X_2 = X$. Since Q_1 is eventually non-retracting, we obtain that $(s \upharpoonright A_1, \aleph_1 \cup [t,\infty) \times X_1) \in \mathscr{F}_{TI}[\![Q_1]\!]$. Since P_1 is refined by Q_1, we have that $(f,(tr \upharpoonright A_1, X_1)) \in \mathscr{F}_{UI}[\![P_1]\!]$.

Now consider a finite set $\{a_1, a_2 \ldots a_n\} = Y \subseteq X_2$. For each a_i there is a corresponding time t_i such that $[t_i,\infty) \times \{a_i\} \subseteq X_2$. Thus there is a time $t_0 = max\{t_i\}$ such that $[t_0,\infty) \times Y \subseteq \aleph_2$. Since P_2 is refined by Q_2, it follows that $(f,(tr \upharpoonright A_2, Y)) \in \mathscr{F}_{UI}[\![P_2]\!]$. This is true for all finite subsets Y of X_2, so by compactness of P_2 we have that $(f,(tr \upharpoonright A_2, X_2)) \in \mathscr{F}_{UI}[\![P_2]\!]$. Hence, $(f,(tr,X)) \in \mathscr{F}_{UI}[\![P_1 \ {}_{A_1}\|_{A_2} P_2]\!]$ as required. □

Compactness is often easy to check, since it will be present in any program not containing infinite non-determinism. Thus any program not containing any infinite choice or infinite-to-one renaming will automatically be compact.

3.1.5. Specification

In the untimed infinite traces model specifications may be considered as consisting of three components, dealing with the failures, divergences and infinite traces. In other words, for any given $S(l,b)$ there are S_f, S_d and S_i such that

$$\begin{aligned} S(l,b) \Leftrightarrow\ & (l = f \wedge b = (tr,X)) \Rightarrow S_f(tr,X) \\ & \wedge (l = d \wedge b = tr) \Rightarrow S_d(tr) \\ & \wedge (l = i \wedge b = u) \Rightarrow S_i(u) \end{aligned}$$

Then a specification $(S_f(tr,X), S_d(tr), S_i(u))$ translates to the timed specification

$$\begin{aligned} S'(s,\aleph) \Leftrightarrow\ & \#s < \infty \wedge [t,\infty) \times X \subseteq \aleph \Rightarrow S_f(strip(s),X) \\ & \wedge \#s = \infty \Rightarrow S_i(strip(s)) \end{aligned}$$

For example, the specification 'deadlock-free' constrains only the possible failure set, with $S_f(tr,X) \Leftrightarrow X \neq \Sigma$. The translation is equivalent to

$$\#s < \infty \Rightarrow \neg\exists t \bullet [t,\infty) \times \Sigma \subseteq X$$

which is the timed version of deadlock-freedom. Thus an untimed verification of deadlock-freedom for a system remains valid under timewise refinement.

The untimed specification of a buffer which passes messages of type M may be given simply as a predicate S_f:

$$\begin{aligned} S_f(tr,X) \Leftrightarrow\ & tr \downarrow out \leqslant tr \downarrow in \\ & \wedge\ tr \downarrow out = tr \downarrow in \Rightarrow X \cap in.M = \{\} \\ & \wedge\ tr \downarrow out < tr \downarrow in \Rightarrow out.M \not\subseteq X \end{aligned}$$

The translation is equivalent to the following timed specification:

$$\begin{aligned} & strip(s) \downarrow out \leqslant strip(s) \downarrow in \\ \wedge\ & strip(s) \downarrow out = strip(s) \downarrow in \Rightarrow \neg\exists t,m \bullet [t,\infty) \times \{in.m\} \subseteq \aleph \\ \wedge\ & strip(s) \downarrow out < strip(s) \downarrow in \Rightarrow \neg\exists t \bullet [t,\infty) \times out.M \subseteq \aleph \end{aligned}$$

which is the specification of a timed buffer.

As an example of an application of the theory, consider Roscoe's first (untimed) buffer law presented in [13], which tells us that the chaining together of two buffers is again a buffer. The chaining operator is defined in terms of parallel, hiding, and renaming (where $swap_{a,b}$ renames channel a to b and vice versa):

$$P_1 \gg P_2 \mathrel{\widehat{=}} (swap_{out,c}(P_1) \ {}_{\{in,c\}}\|_{\{out,c\}}\ swap_{in,c}(P_2)) \setminus c$$

However, this law does not hold in general in the timed model. As we have seen, B_1 and B_2 might fail to agree on a time to synchronise on their common internal channel, resulting in their combination refusing ever to output.

In order to establish conditions under which the law does hold, we will make use of the fact that untimed buffers are compact (since if some input can be refused, then so too can all possible inputs), and also of the fact that the refinement relations in this paper are complete, which yields that every timed buffer is a refinement of some untimed buffer. We may then obtain conditions under which a chain of buffers again yields a buffer.

For example, if every timed buffer B_i is eventually non-retracting on input, then the chain $B_1 \gg B_2 \gg \cdots \gg B_n$ is again a buffer, eventually non-retracting on input. This follows from the fact that each B_i is a refinement of some untimed buffer A_i (by completeness of the refinement relation); that we have a condition which may be applied at every step of building up the chain to ensure that the timed chain refined the untimed chain (in the general parallel case, we require only non-retraction on the interface); and that the chain $A_1 \gg A_2 \gg \cdots \gg A_n$ is an untimed buffer (from Roscoe's law), from which it follows that any refinement of it is a timed buffer. A similar result holds if each B_i is non-retracting on output; or if odd (or even) numbered buffers are non-retracting on both input and output. It follows that the combination $B_1 \gg COPY \gg B_2 \cdots COPY \gg B_n$ is a buffer, for any timed buffers B_i.

4. An operational characterisation

A more immediately intuitive semantic approach often employed in the theory of process algebra is that of operational semantics: processes are defined in terms of transitions that they may perform and subsequent states that may be reached. Within this framework, equivalence between processes may be characterised in terms of bisimulation relations [19], or by means of equivalence under some notion of testing [12].

In the testing approach, a test is defined to be a process T which also has the capacity to perform a special success event ω, which is considered to be distinct from the set of synchronisation events Σ. An execution of a process P is a maximal (finite or infinite) sequence of transitions starting from P. Then we say that P may T if there is some execution of $(P \| T) \setminus \Sigma$ which passes through a state from which ω is a possible transition; and P must T if every execution of $(P \| T) \setminus \Sigma$ passes through such a state. Then P is equivalent to Q under may testing if for any test T, P may $T \Leftrightarrow Q$ may T; and P and Q are equivalent under must testing if P must $T \Leftrightarrow Q$ must T for any test T.

An operational semantics has been given for CSP in [4, 24]. Transitions are given as $P \xrightarrow{\mu} P'$, indicating that a process P may perform a μ event (i.e. an internal or visible event) and then behave as P'. In this section we will subscript the transition with a u to indicate that this is an untimed transition. Equivalence in the untimed traces model $\mathcal{M}_{UT}$ is exactly the same as equivalence under may testing using the transitions given in [24]; and equivalence in the untimed infinite traces model $\mathcal{M}_{UI}$ is exactly the same as equivalence under must testing using those transitions. More details may be found in [12, 4, 24]. The important properties from our point of view is that each trace of P predicted by the traces model corresponds to an execution of P in

which that sequence of visible events is performed (as well as possibly some internal events); that any divergence corresponds to an execution in which some prefix of the divergent trace is performed, followed by an infinite sequence of internal τ steps; any failure (tr, X) corresponds either to a divergence (i.e. an infinite sequence of τ steps after some prefix of the trace) or to an execution in which the entire sequence tr of events is performed, and a state is reached from which no internal progress can be made, and from which no event in the refusal set X is possible; and for every infinite trace u there is an execution that either diverges after some prefix of u or performs the entire sequence of events u. And conversely, any execution given by the operational semantics is recorded appropriately in the denotational semantics.

An operational semantics has also been provided for timed CSP in [27, 29], where processes may undergo timed transitions: $P \xrightarrow{(t,\mu)} P'$ indicates that the process P may perform event μ at time t, and subsequently behave as P'. We will subscript timed transitions with t to distinguish them from untimed transitions. Evolutions, or time passing transitions, were also provided in the operational semantics. Equivalence in the infinite timed failures model $\mathcal{M}_{TI}$ is the same as equivalence under must testing using the transitions given in [29]. Again, timed failures $(s, \aleph)$ are present in the denotational semantics of a process P precisely when there is some execution of P in which events are performed at the times recorded in s, passing through states in which the events recorded in the refusal set $\aleph$ were not possible.

Both may and must forms of testing can be defined for both untimed and timed CSP. We will say $P \underline{\text{may}}_u T$ if some execution of $(P \| T) \backslash \Sigma$ in terms of untimed $\rightarrow_u$ transitions passes through a state in which ω is possible; and we will use $P \underline{\text{must}}_u T$, $P \underline{\text{may}}_t T$ and $P \underline{\text{must}}_t T$ in a similar fashion.

We may interpret a timed program Q also at the untimed level. Given a timed program Q, we may define an untimed program $\Theta(Q)$ whose untimed transitions are derived from the timed ones:

$$\frac{Q \xrightarrow{(t,\mu)}_t Q'}{\Theta(Q) \xrightarrow{\mu}_u \Theta(Q')}$$

Then $\mathscr{F}_{UT}[\![\Theta(Q)]\!]$, $\mathscr{F}_{UD}[\![\Theta(Q)]\!]$, $\mathscr{F}_{UF}[\![\Theta(Q)]\!]$, and $\mathscr{F}_I[\![\Theta(Q)]\!]$ can all be defined using the resulting transition system.

In the traces model, $P \sqsubseteq_{UT} Q$ is true exactly when $\forall T \bullet (Q \underline{\text{may}} T \Rightarrow P \underline{\text{may}} T)$. By analogy, we may characterise an operational version of timed refinement, where if Q may pass a timed test T, then P may pass the same T considered as an untimed test.

Definition 4.1. $P \underset{\sim_t}{\sqsubseteq} Q$ is defined by

$$P \underset{\sim_t}{\sqsubseteq} Q \Leftrightarrow \forall T \bullet Q \underline{\text{may}}_t T \Rightarrow P \underline{\text{may}}_u \Theta(T)$$

It turns out that this notion of refinement is the same as the denotational version of traces refinement.

Theorem 4.2. $P \underset{\sim_t}{\sqsubseteq} Q \Leftrightarrow P \sqsubseteq_{UT\mathscr{R}_{TI}} Q$

Proof. "⇒" Assume $P \not\sqsubseteq_{UT\mathscr{R}_{TI}} Q$. Then there is some trace s of Q such that $strip(s)$ is not a trace of P. Let $s = \langle (t_1, a_1), \ldots, (t_n, a_n) \rangle$. Then define the test T by

$$T = a_1 \rightarrow \cdots \rightarrow a_n \rightarrow \omega \rightarrow Stop$$

Since the timed operational semantics are equivalent to the denotational semantics, there is some execution of Q giving rise to trace s, so there is some execution of $(Q \| T) \backslash \Sigma$ which reaches a state in which T can perform ω. However, there is no such execution of $(P \| \Theta(T)) \backslash \Sigma$, since if there were then this would correspond to P performing the events in $strip(s)$, which would mean that $strip(s)$ is a trace of P, yielding a contradiction. Thus Q $\underline{\text{may}}_t$ T but $\neg(P$ $\underline{\text{may}}_u$ $T)$, and so $\neg(P \underset{\sim_t}{\sqsubseteq} Q)$

"⇐" Assume $P \sqsubseteq_{UT\mathscr{R}_{TI}} Q$, and consider a test T for which Q $\underline{\text{may}}_t$ T. Then there is some execution of $(Q \| T) \backslash \Sigma$ which leads to a state in which $\overset{\omega}{\rightarrow}_t$ is possible. The contribution of Q to this execution corresponds to some timed trace s. Then $strip(s)$ is a trace of P, so P has some execution giving rise to $strip(s)$. Thus $(P \| \Theta(T)) \backslash \Sigma$ has an execution which takes $\Theta(T)$ through states corresponding to the timed states that T passes through in the successful execution of $(Q \| T) \backslash \Sigma$, and so it reaches a state in which an $\overset{\omega}{\rightarrow}_u$ transition is possible. Thus, P $\underline{\text{may}}_u$ T. □

In the failures/divergences model, $P \sqsubseteq_{UI} Q$ is equivalent to P $\underline{\text{must}}$ $T \Rightarrow Q$ $\underline{\text{must}}$ T for any T. Again by analogy, we characterise an operational version of timed refinement:

Definition 4.3. $P \underset{\sim_f}{\sqsubseteq} Q$ is defined by

$$P \underset{\sim_f}{\sqsubseteq} Q \;\Leftrightarrow\; \forall T \bullet P \;\underline{\text{must}}_u\; \Theta(T) \Rightarrow Q \;\underline{\text{must}}_t\; T$$

This formulation of refinement is equivalent to the denotational version of failures refinement.

Theorem 4.4. $P \underset{\sim_f}{\sqsubseteq} Q \Leftrightarrow P \sqsubseteq_{UI\mathscr{R}_{TI}} Q$

Proof. "⇒" If $P \not\sqsubseteq_{UI\mathscr{R}_{TI}} Q$ then either (1) there is some $(s, [t', \infty) \times X) \in \mathscr{F}_{TI}[\![Q]\!]$ with $(f, (strip(s), X)) \notin \mathscr{F}_{UI}[\![P]\!]$, or (2) there is some infinite trace s such that $(s, \{\}) \in \mathscr{F}_{TI}[\![Q]\!]$ and $strip(s) \notin \mathscr{F}_{UI}[\![P]\!]$.

1. Let $s = \langle (t_1, a_1), \ldots, (t_n, a_n) \rangle$, and let $t_0 = 0$. Assume, without loss of generality, that $t' \geqslant t_n$. Define

$$T_i = Wait[(t_i - t_{i-1})]; ((a_i \rightarrow T_{i+1}) \overset{0}{\triangleright} \omega \rightarrow Stop)) \quad 0 < i \leqslant n$$

$$T_{n+1} = Wait[(t' - t_n)]; x : X \rightarrow \omega \rightarrow Stop$$

If $(P\|\Theta(T_1))\backslash\Sigma$ has an execution that is not successful, then the contribution from P must correspond to the failure $(strip(s), X)$, yielding a contradiction. Thus P $\underline{\text{must}}_u$ T_1. On the other hand, Q has an execution corresponding to $(s, [t', \infty) \times X)$, and so $(Q\|T_1)\backslash\Sigma$ does have an unsuccessful execution, thus $\neg(Q$ $\underline{\text{must}}_t$ $T_1)$.

2. Let $s = \langle (t_1, a_1), \ldots, (t_i, a_i), \ldots \rangle$. Then let the trace during an interval $[n, n+1)$ be given by $\langle (t_{n,1}, a_{n,1}), \ldots, (t_{n,m}, a_{n,m}) \rangle$. This must be finite for any interval, since the trace s is finitely variable, i.e. its restriction to any finite interval is finite. Define

$$T_{n,i} = Wait[(t_{n,i} - t_{n,i-1})]; \\ ((a_n, i \to T_{n,i+1}) \overset{0}{\rhd} \omega \to Stop) \qquad 0 < i \leqslant m$$

$$T_{n,m+1} = Stop$$

$$T_n = T_{n,1} \,|||\, Wait[1]; T_{n+1}$$

Each $T_{n,i}$ is well-defined, so each of the equations for the T_i is 1-guarded.) Then if $(P\|\Theta(T_0))\backslash\Sigma$ has an unsuccessful execution, the contribution of P must correspond to $strip(s)$, yielding a contradiction; thus P $\underline{\text{must}}_u$ $\Theta(T_0)$. However, $(Q\|T_0)\backslash\Sigma$ does have an unsuccessful execution, driven by an execution of Q corresponding to s. Thus, $\neg(Q$ $\underline{\text{must}}_t$ $T_0)$.

"$\Leftarrow$" Assume that $P \sqsubseteq_{UI\mathscr{R}_{TI}} Q$, and that $\neg(Q$ $\underline{\text{must}}_t$ $T)$. It will be enough to prove that $\neg(P$ $\underline{\text{must}}_u$ $\Theta(T))$. Consider an unsuccessful execution of $(Q\|T)\backslash\Sigma$. There are a number of possibilities; we consider the events that were internalised by the $\backslash\Sigma$ abstraction:

- *$Q\|T$ performs infinitely many events from Σ.* Then there is a corresponding infinite trace s of both Q and T. Since $P \sqsubseteq_{UI\mathscr{R}_{TI}} Q$, the trace $strip(s)$ is an infinite trace of P. If P diverges at some point along $strip(s)$, then this will give rise to an unsuccessful execution of $(P\|T)\backslash\Sigma$. Otherwise there is an infinite untimed execution of $\Theta(T)$ performing the same events, and passing through corresponding states to those reached in the timed execution. Hence there is an infinite execution of $(P\|\Theta(T))\backslash\Sigma$ where ω is not possible in any state (since the possibility of ω depends purely on the state reached by $\Theta(T)$), and so $\neg(P$ $\underline{\text{must}}_u$ $\Theta(T))$.
- *$Q\|T$ performs finitely many events from Σ:*
  - If T performs infinitely many timed τ transitions, then $\Theta(T)$ may perform infinitely many untimed ones, passing through corresponding states, so if P does not diverge (which itself leads to an unsuccessful execution) then this will yield an unsuccessful execution of $(P\|\Theta(T))\backslash\Sigma$.
  - If T performs finitely many τ actions, then it will come to arrive in a final state T'. Any events that T' is able to perform are blocked by Q for all time, and so P (if it does not diverge) may reach a stable state P' in which none of those events are possible. Also, $\Theta(T)$ may by untimed transitions reach a state $\Theta(T')$, which is also unable to perform those events that T' was unable to perform, and so $P'\|\Theta(T')$ will be unable to progress. Thus the execution from $P\|\Theta(T)$ to $P'\|\Theta(T')$ is maximal, and unsuccessful. □

These results may also be used to establish that any untimed description of a system in CSP is refined by the same description considered as a timed description.

We first provide an untimed operational semantics for the timeout operator. The timeout may always be resolved by its left-hand argument performing a visible action, but any internal progress made by that argument does not resolve the timeout:

$$\frac{P \xrightarrow{a}_u P'}{P \overset{t}{\rhd} P'' \xrightarrow{a}_u P'} \qquad \frac{P \xrightarrow{\tau}_u P'}{P \overset{t}{\rhd} P'' \xrightarrow{\tau}_u P' \overset{t}{\rhd} P''}$$

Furthermore, the timeout may occur:

$$\frac{}{P \overset{t}{\rhd} P'' \xrightarrow{\tau}_u P''}$$

Lemma 4.5. *Any timed process P has $\Theta(P) \sqsubseteq_{\mathcal{TT}} P$ and $\Theta(P) \sqsubseteq_{\mathcal{UT}} P$.*

Proof (*sketch*). This follows from the above-mentioned equivalence of the denotational and operational semantics in both the untimed cases and the timed case. In the first case, if there is a timed trace s of P, then there is some execution of P which gives rise to this trace. But then every step of this execution can be matched by an untimed step, so there is an equivalent untimed execution of $\Theta(P)$, which corresponds to the trace $strip(s)$. Since, the untimed operational and denotational semantics are equivalent, the trace $strip(s)$ appears in the trace set of $\Theta(P)$.

In the case of failures refinement, similar reasoning shows that infinite timed traces will be matched by infinite untimed ones; and a timed failure $(s, [t, \infty) \times X)$ with finite trace s will correspond to some execution of P. After the trace s has been performed there are two possibilities. The execution may contain an infinite sequence of internal events; these can be matched by $\Theta(P)$, leading to a divergence and the inclusion of $(f, (strip(s), X))$ as a failure of $\Theta(P)$. The other possibility is that a final state is reached from which no event in X, or any further internal progress, is possible (since X is refused from that point onwards); in this case a corresponding untimed state in which X may be refused is reachable from $\Theta(P)$ by means of a corresponding execution, and the failure $(f, (strip(s), X))$ again appears as a failure of $\Theta(P)$. □

Definition 4.6. Define relation R to be a *failures simulation* if whenever $R(P_1, P_2)$, then

1. If $P_2 \xrightarrow{\mu}_u P_2'$ then there is some P_1' such that $P_1 \xrightarrow{\mu}_u P_1'$ and $R(P_1', P_2')$.
2. If $P_1 \xrightarrow{\mu}_u$ then there are processes P_1', P_2', such that $P_1 \xrightarrow{\mu}_u P_1'$ and $P_2 \xrightarrow{\mu}_u P_2'$ and $R(P_1', P_2')$.

We say that P_1 *failures simulates* P_2 if there is some failures simulation between them.

Lemma 4.7. *If P_1 failures simulates P_2 then $P_1 \sqsubseteq_{UT} P_2$.*

Proof. It follows from the definition of failures simulation that any trace, infinite trace, or divergent trace of P_2 must also be a trace, infinite trace, or divergent trace, respectively, of P_1. Furthermore, any refusal of P_2 (not arising from divergence) is also a refusal of P_1, since if $R(P_1, P_2)$ and $\neg P_2 \xrightarrow{\mu}_u$ then $\neg P_1 \xrightarrow{\mu}_u$; it follows that any failure of P_2 is a failure of P_1. □

Lemma 4.8. *Any CSP process P has P failures simulates $\Theta(P)$.*

Proof. Let the relation R hold between two programs P and $\Theta(Q)$ if P and Q are syntactically identical up to the values of timeouts. A straightforward structural induction on P shows that R is a failures simulation.

As an illustration we will establish the case of the timeout operator.

Assume that P and Q are syntactically identical up to the values of timeouts. If $P = P_1 \overset{t}{\rhd} P_2$, then $Q = Q_1 \overset{t'}{\rhd} Q_2$, where P_i is syntactically identical to Q_i up to the values of timeouts for $i = 1, 2$.

The operational rules required to establish this case are the following:

$$\frac{Q \xrightarrow{(t',a)}_t Q'}{Q \overset{t}{\rhd} Q'' \xrightarrow{(t',a)}_t Q'} [t' \leqslant t] \qquad \frac{Q \xrightarrow{(t'\tau)}_t Q'}{Q \overset{t}{\rhd} Q'' \xrightarrow{(t',\tau)}_t Q' \overset{t-t'}{\rhd} Q''} [t' \leqslant t]$$

Furthermore, the timeout may occur if Q can evolve for t units of time:

$$\frac{Q \overset{t}{\rightsquigarrow}}{Q \overset{t}{\rhd} Q'' \xrightarrow{(t,\tau)}_t Q''}$$

There are two clauses that define failures simulation.

Clause 1. If $\Theta(Q) \xrightarrow{\mu}_u R$ then $Q \xrightarrow{(t',\mu)}_t Q'$ for some t', Q' by the definition of Θ. There are three possibilities to be considered, one for each of the above operational rules. We will consider only the second; the other cases are entirely similar. In this case we have $Q_1 \xrightarrow{(t',\tau)}_t Q_1'$. The inductive hypothesis states that $P_1 \xrightarrow{\tau}_u P_1'$ with P_1' identical to Q_1' up to the values of timeout, so it follows that $P_1' \overset{t}{\rhd} P_2$ is identical to $Q_1' \overset{t-t'}{\rhd} Q_2$ up to the value of timeouts, as required.

Clause 2. If $P \xrightarrow{\mu}_u$ then there are two possibilities: either $\mu = \tau$ or $\mu \neq \tau$. In the latter case the proof is straightforward. In the case where $\mu = \tau$, it is a result from [29] that any timed process can either evolve or perform a τ transition: either (1) $Q_1 \overset{t}{\rightsquigarrow}$ or (2) $Q_1 \xrightarrow{(t',\tau)}_t Q_1'$ for some $t' \leqslant t$.

In case (1), the third operational rule is applicable, and so $Q \xrightarrow{(t,\tau)}_t Q_2$. Hence $\Theta(Q) \xrightarrow{\tau}_u \Theta(Q_2)$. The third untimed rule for timeout has that $P \xrightarrow{\tau}_u P_2$, and by the inductive hypothesis on the relationship between P_2 and Q_2 it follows that $R(P_2, \Theta(Q_2))$ as required.

In case (2), the inductive hypothesis for P_1 and Q_1 states that there are processes P_1' and Q_1' such that $P_1 \xrightarrow{\tau}_u P_1'$ and $\Theta(Q_1) \xrightarrow{\tau} Q_1'$ with $R(P_1', Q_1')$. Hence there are processes

$P_1' \overset{t}{\rhd} P_2$ and $Q_1' \overset{t-t'}{\rhd} \Theta(Q_2)$ which are related by R, and for which $P \xrightarrow{\tau}_u P_1' \overset{t}{\rhd} P_2$ and $\Theta(Q) \xrightarrow{\tau}_u Q_1' \overset{t-t'}{\rhd} \Theta(Q_2)$, as required.

Thus, the case is established.

Hence, P failures simulates $\Theta(Q)$ whenever P and Q are identical up to the values of timeouts. The lemma follows from the special case $P = Q$. □

Theorem 4.9. *Any process P has $P \sqsubseteq_{UTM_{TI}} P$ and $P \sqsubseteq_{UI M_{TI}} P$.*

Proof. Lemmas 4.5, 4.7, 4.8 yield that

$$P \sqsubseteq_{UI} \Theta(P) \sqsubseteq_{UI M_{TI}} P$$

and Lemma 1.4 yields that

$$P \sqsubseteq_{UI M_{TI}} P$$

as required. □

5. Examples

5.1. A stop and wait protocol

The well-known alternating bit protocol is a useful common example, since it has been treated by so many different formalisms that it provides a means of comparing and contrasting them. We will use it here simply to illustrate some of the techniques presented earlier.

The untimed alternating bit protocol consists of a sender and receiver communicating over two lossy channels. The nature of a generic lossy channel may be specified at the untimed level using the infinite traces model. The specification SM on a medium $M_{in,out}$ with input in and output out consists of three parts, two on the trace refusal observations, and one on the infinite traces:

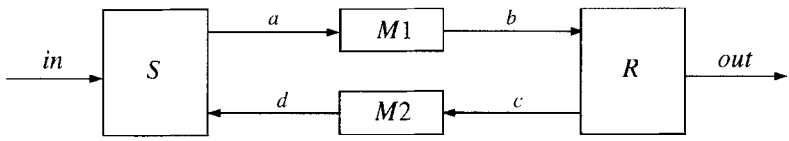
$$\begin{aligned} SM1(tr, X) &\quad tr \downarrow out \preccurlyeq tr \downarrow in \\ SM2(tr, X) &\quad out.M \not\subseteq X \vee in.M \cap X = \{\} \\ SM3(u) &\quad \#(u \upharpoonright in) = \infty \Rightarrow \#(u \upharpoonright out) = \infty \end{aligned}$$

$SM1(tr, X)$ simply states that the sequence of messages passed on channel out should be a (not necessarily contiguous) subsequence of those passed on channel in, so messages may be lost but not corrupted; $SM2(tr, X)$ states that at least one of input and output should not be refused (where X is the refusal set); and $SM3(u)$ is a fairness condition that requires that output should not be lost infinitely often.

The requirement we have of the entire system is that it should behave as a (one-place) buffer. Our specification is

$$\begin{aligned} SPEC(tr,X) = \quad & tr \downarrow out \leqslant_1 tr \downarrow in \\ & \wedge\ tr \downarrow out = tr \downarrow in \Rightarrow in.M \cap X = \{\} \\ & \wedge\ tr \downarrow out < tr \downarrow in \Rightarrow out.M \notin X \end{aligned}$$

The network used is pictured as follows:



The basic idea of the protocol is to add an extra bit to each of the messages sent along the lossy channels which alternates between 0 and 1. The sending process sends multiple copies of each message until it receives an acknowledgement. As soon as the receiving process gets a new message it sends acknowledgements of it until the next message arrives. The two ends can always spot a new message or acknowledgement because of the alternating bit.

The two media are described as $M1 = M_{a,b}$ and $M2 = M_{c,d}$, passing messages from a to b, and from c to d.

This strategy may be captured by the following CSP descriptions of the sender S and the receiver R. We set $R = R(0)$ and $S = S(0)$, where for $s \in \{0,1\}$ and x in the set of messages M we define

$$S(s) = in?x \rightarrow S'(s,x)$$

$$\begin{aligned} S'(s,x) = \quad & a!(s,x) \rightarrow S'(s,x) \\ & \Box\ d?s \rightarrow S(\bar{s}) \\ & \Box\ d?\bar{s} \rightarrow a!(s,x) \rightarrow S'(s,x) \end{aligned}$$

$$\begin{aligned} R(s) = \quad & b?(s,x) \rightarrow out!x \rightarrow c!s \rightarrow R(\bar{s}) \\ & \Box\ b?(\bar{s},x) \rightarrow c!\bar{s} \rightarrow R(s) \end{aligned}$$

The entire network consists of the parallel combination of the sender and receiver together with the two media; and the channels a,b,c, and d are all made internal.

$$NETWORK = ((S \,\|_{\{in,a,d\}}\, \|_{\{out,b,c\}}\, R)_\Sigma \,\|_{\{a,b,c,d\}}\, (M1 \,\|_{\{a,b\}}\, \|_{\{c,d\}}\, M2)) \setminus \{a,b,c,d\}$$

The network is considered as the parallel combination of the protocol and the media.

An analysis at the untimed level establishes that the system is livelock-free, essentially because of the fairness of the media which cannot lose an infinite sequence of messages. It is also deadlock-free: if S cannot make progress, then it must be waiting for both media, which must therefore both be ready to interact with R, and so R is able to make progress. Finally, it is straightforward to show that it is functionally equivalent to a one-place buffer.

Timed descriptions of the alternating bit protocol commonly employ a timeout in the description of the sender process, since the intention is that the sender should wait for an acknowledgement and then retransmit a message if this does not arrive within a certain interval. But in fact there is no need to withdraw the capability of receiving a message on the acknowledgement channel simply because a retransmission has been enabled, and so at the untimed level this behaviour may be modelled as a choice.

To provide a timed refinement of the protocol, we wish to preserve correctness of the system. The most general form of correctness that could be preserved by a timewise refinement would be for timed versions of the media to meet simply the translations of the untimed specifications with no further constraints. Thus, we prefer not to impose the restriction on the media that they are non-retracting.

A timed version TS of the sender process may be obtained simply by including a delay t before retransmission of a message. The length of this delay will be influenced by such factors as the length of time before an acknowledgement would be expected to arrive, and the reluctance to send unnecessary messages. The timed receiver process TR still behaves sequentially, and has no time-critical behaviour.

Some small delays ε are introduced to ensure that the recursive loops are time-guarded. (These play the role of the original δ delay enforced by event prefix in earlier versions of timed CSP [23]).

$$
\begin{aligned}
TS(s) &= in?x \rightarrow TS'(s,x) \\
TS'(s,x) &= Wait[t]; a!(s,x) \rightarrow TS'(s,x) \\
&\quad \Box\ d?s \xrightarrow{\varepsilon} TS(\bar{s}) \\
&\quad \Box\ d?\bar{s} \xrightarrow{\varepsilon} a!(s,x) \rightarrow TS'(s,x) \\
TR(s) &= b?(s,x) \xrightarrow{\varepsilon} out!x \rightarrow c!s \rightarrow TR(\bar{s}) \\
&\quad \Box\ b?(\bar{s},x) \xrightarrow{\varepsilon} c!\bar{s} \rightarrow TR(s)
\end{aligned}
$$

Given two timed media $TM1$ and $TM2$ that meet the timed translation of SM, by completeness there are two untimed media $M1$ and $M2$ which meet SM and which are refined by $TM1$ and $TM2$. Then $M1_{\{a,b\}} \parallel_{\{c,d\}} M2 \sqsubseteq_{UTT} TM1_{\{a,b\}} \parallel_{\{c,d\}} TM2$ by Lemma 3.10, since the intersection of the interface sets is empty, so both $TM1$ and $TM2$ are trivially non-retracting on it. Also, by Theorem 3.5, and since delays may be introduced into an untimed description to produce a timed refinement, and the intersection of the two interface sets is empty, we have that $S_{\{in,a,d\}} \parallel_{\{out,b,c\}} R \sqsubseteq_{UTT} TS_{\{in,a,d\}} \parallel_{\{out,b,c\}} TR$. Furthermore, both the sender and the receiver are non-retracting and prompt, and so $TS_{\{in,a,d\}} \parallel_{\{out,b,c\}} TR$ is also non-retracting and prompt. Thus the timed network

$$
\begin{aligned}
&TNETWORK \\
&= ((TS_{\{in,a,d\}} \parallel_{\{out,b,c\}} TR)_{\Sigma} \parallel_{\{a,b,c,d\}} (TM1_{\{a,b\}} \parallel_{\{c,d\}} TM2)) \setminus \{a,b,c,d\}
\end{aligned}
$$

is a timewise refinement of the untimed network, and so it must be a one-place buffer. Thus the functional correctness of the timed network may be deduced from an untimed analysis.

Of course, to do an analysis of the timing behaviour of the network it would be necessary to use the full power of the timed model. To consider the maximum time between input and output it is necessary to know for how long it is necessary to input messages into the media before output can be guaranteed; and to optimise the value of the timeout t it is necessary to know the expected delay in the media of a successfully transmitted message. The technique of timewise refinement cannot contribute to these concerns; its role is rather to complement them by allowing the appropriate use of more abstract methods for some analysis of aspects of a system's behaviour, even when other aspects require the use of the more complicated timed models.

5.2. A railroad crossing

This example was originally presented in [15]. It gives an extremely simple model of a railroad crossing which is nevertheless complex enough to illustrate a number of aspects of the modelling and verification of timed systems.

The system is described as consisting of three components: a train, a gate, and a gate controller. The gate should be up to allow traffic to pass when no train is approaching, but should be lowered to obstruct traffic when a train is close to reaching the crossing. It is the task of the controller to monitor the approach of a train, and to instruct the gate to be lowered within the appropriate time. The train is modelled at a high level of abstraction: the only relevant aspects of the train's behaviour are when it is nearing the crossing, when it is entering it, when it is leaving it; and the delays between these events.

A number of safety conditions are formulated. These require the gate to be down when the train enters the crossing; the gate not to change state for ten time units before the train enters the crossing; and the train to have left the crossing by the time the gate goes up. We also require the liveness property that the crossing is deadlock-free. A system that deadlocked with the gate down would meet the safety conditions, but would not be satisfactory.

We begin with an untimed description and analysis of the system, to investigate which of these properties may be verified at the untimed level. We keep the process descriptions as simple as possible, including only those events that are relevant to consideration of these properties.

The gate controller *Controller* receives two types of signal from the crossing sensors: *near.ind*, which informs the controller that the train is approaching, and *out.ind*, which indicates that the train has left the crossing. It sends two types of signal to the crossing gate mechanism: *down.command*, and *up.command*, which instruct the gate to go down and up respectively. These four events form the alphabet C of the controller.

The gate, modelled by *Gate*, responds to the commands sent by the controller. We include additional events *up* and *down* to model the position of the gate. These two events, together with *up.command* and *down.command* form the alphabet G of the gate.

This leaves us with the following description of the crossing mechanism:

$$Crossing = Controller \ {}_C\|_G \ Gate$$

The controller responds to sensory inputs by issuing gate command signals:

$$\begin{aligned} Controller = {} & near.ind \rightarrow down.command \rightarrow Controller \\ & \sqsubset \\ & out.ind \rightarrow up.command \rightarrow Controller \end{aligned}$$

The gate process responds to the controller's signals by raising and lowering the gate.

$$\begin{aligned} Gate = {} & down.command \rightarrow down \rightarrow Gate \\ & \Box \\ & up.command \rightarrow up \rightarrow Gate \end{aligned}$$

To reason about the behaviour of the system as a train approaches and reaches the crossing, we model the effect of such a happening via the crossing sensors. The train triggers the sensors by means of the *near.ind* and *out.ind* events. The events *train.near*, *enter.crossing* and *leave.crossing* model respectively the situations where the train is close to the crossing, the train enters the crossing, and the train leaves the crossing. These five events are all that are required for the sake of this analysis: they form the alphabet T of the train.

We will use the process *Train* to model the approach of the train, and its effect upon the crossing system.

$$\begin{aligned} Train = {} & train.near \rightarrow near.ind \rightarrow enter.crossing \rightarrow \\ & leave.crossing \rightarrow out.ind \rightarrow Train \end{aligned}$$

The crossing system, in conjunction with the train, is described as follows:

$$System = Train \ {}_T\|_{C \cup G} \ Crossing$$

We are now able to express the properties we gave earlier in terms of the events we have chosen to model the system.

If the train enters the crossing, then the gate should have gone down more recently than it went up:

$$Safety1(tr) = last(tr) = enter.crossing \Rightarrow last(tr \upharpoonright \{down, up\}) = down.$$

If the train enters the crossing at time t, then no *down* or *up* events should have occurred in the preceding ten time units; the projection of the trace to those events over that interval is empty:

$$Safety2(s, \aleph) = \langle (t, enter.crossing) \rangle \in s \Rightarrow (s \upharpoonright \{down, up\} \uparrow [t - 10, t]) = \langle \rangle$$

If the gate goes up, then the train must have left the crossing more recently than it entered it:

$$\begin{aligned} Safety3(tr) = {} & last(tr) = up \Rightarrow \\ & last(tr \upharpoonright \{enter.crossing, leave.crossing\}) = leave.crossing \end{aligned}$$

Finally, the system must be deadlock-free.

$$Liveness1(tr, \aleph) = \text{deadlock-freedom}$$

Each of these properties has been expressed at the highest possible level of abstraction. In each case, the simplest model has been used to capture the required property. Safety properties 1 and 3 are expressible in the untimed traces model. Safety property 2 concerns explicit timing issues, so the timed model is required in order to express it. Deadlock-freedom is expressible using the untimed failures model.

Safety properties 1 and 3, and the liveness property, are candidates for being established by the untimed system description. They may be established by use of algebraic laws, or by use of proof rules. An alternative approach would be to use model-checking for these properties directly [25]. The states of the system are shown in Fig. 4 (where internal transitions corresponding to unwinding of recursions have been elided). Examination of the diagram reveals that at any point where *up* is possible, there must have been a *leave.crossing* event more recently than an *enter.crossing* event. Thus *Safety*3 is satisfied. On the other hand, there are *enter.crossing* transitions where *up* is more recent than *down*, showing that in fact *Safety*1 is not satisfied. Even though it is expressible as an untimed requirement, it turns out that its validity rests upon timing properties of the system (in particular, that the gate goes down in less time than it takes for the train to reach the crossing). Finally, every state has some transition out of it, so the system is deadlock-free, meeting the liveness requirement. Both *Safety*3 and *Liveness*1 are easily checked by the Failures Divergences Refinement checker FDR [10].

Timewise refinement allows timing information to be added to the description of the system while preserving the properties already established. We firstly include the timing information we have about the train: that it takes at least 5 min from triggering the *near.ind* sensor to reach the crossing; and that it takes at least 20 s to get across the crossing.

$$\begin{aligned} TTrain = train.near \rightarrow near.ind \xrightarrow{300} enter.crossing \xrightarrow{20} \\ leave.crossing \rightarrow out.ind \rightarrow TTrain \end{aligned}$$

The controller takes a negligible amount of time ε from receiving a signal from a sensor to relaying the corresponding instruction to the gate.

$$\begin{aligned} TController = near.ind \xrightarrow{\varepsilon} down.command \rightarrow TController \\ \Box \\ out.ind \xrightarrow{\varepsilon} up.command \rightarrow TController \end{aligned}$$

The timed gate process *TGate* process takes a non-negligible amount of time to get the gate into position following an instruction:

$$\begin{aligned} TGate = down.command \xrightarrow{100} down \rightarrow TGate \\ \Box \\ up.command \xrightarrow{100} up \rightarrow TGate \end{aligned}$$

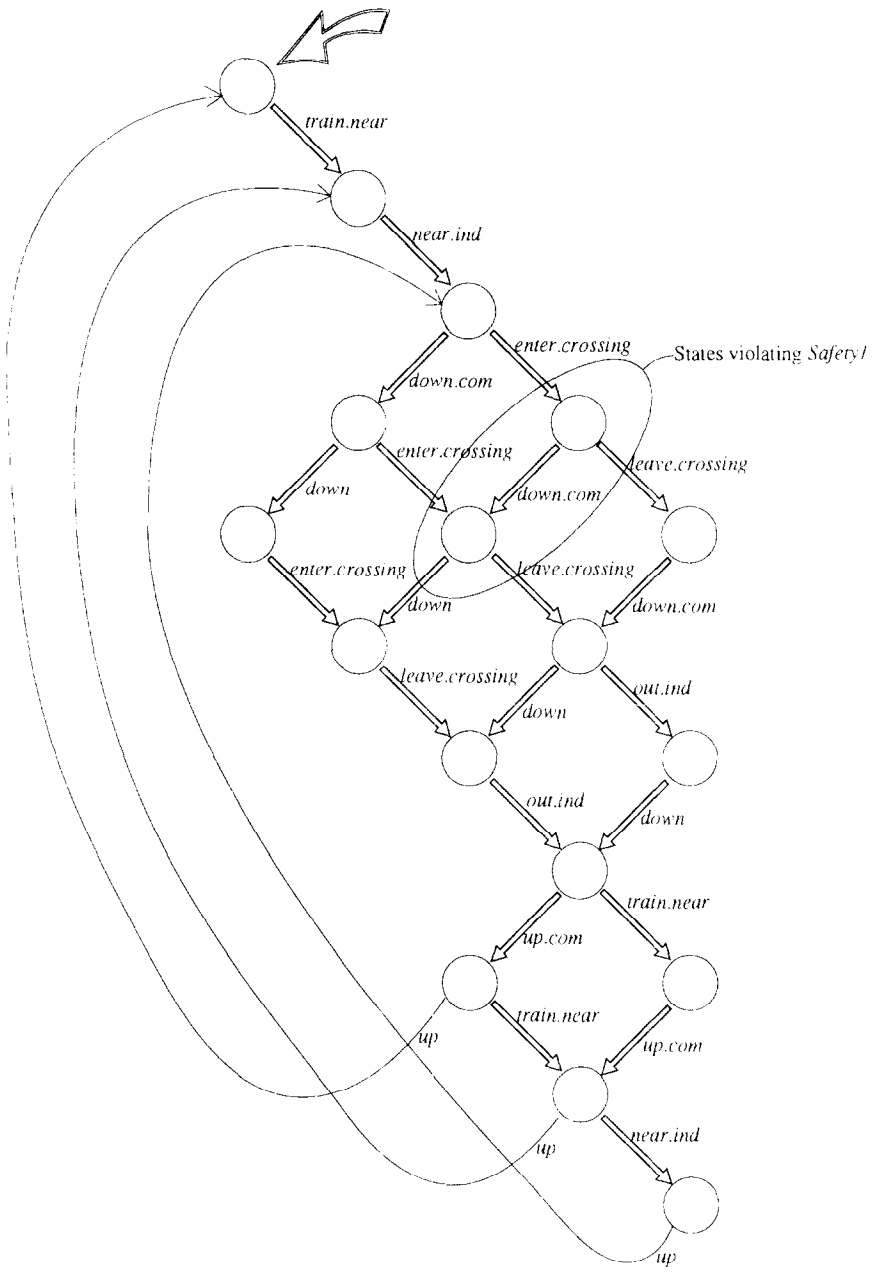


Fig. 4. States and transitions for an untimed railway crossing.

However, this is still considerably less time than it takes for the train to reach the crossing, so the timed description is sufficiently detailed to establish *Safety*1, under the additional assumption that the events *up* and *down* are entirely under the control of the *Gate*. For further discussion of environmental assumptions, see [30]. This environmental

assumption is captured as $[0,\infty) \times \{up, down\} \subseteq \aleph$, and so the specification met by the system is

$$[0,\infty) \times \{up, down\} \subseteq \aleph \Rightarrow Safety1$$

The timed description is also sufficient to establish *Safety*2.

The timed process descriptions are simply the untimed process descriptions with times added. The results of Section 3.1 guarantee that the timed system trace refines the untimed one, ensuring that the timed system meets *Safety*3.

Finally, the three component timed processes are all non-retracting. Lemma 3.8 ensures that the system consisting of their parallel combination is a failures refinement of the untimed system, so it retains the untimed property *Liveness*1.

6. Discussion

We have seen how verifications of specifications can be mapped up the CSP hierarchy of models, and also an example of how general laws might be translated. Other properties (such as deterministic or compact) do not translate in general. For example, the deterministic untimed process $a \rightarrow Stop$ is refined by the non-deterministic timed process $a \rightarrow Stop \sqcap Wait_5; a \rightarrow Stop$, which can perform or refuse to perform a at time 2.

6.1. Comparisons

There has also been some work in this area in the contexts of timed CCS and of timed ACP. Larsen and Yi [14] have proposed a notion of *time-abstracting* bisimulation, which specifies when timed processes are equivalent modulo timing behaviour. Thus one process may be used to specify simply the functional behaviour of a system by requiring that any proposed implementation should be time-abstracting bisimilar to it. They prove that time-abstracting equivalence is decidable for a timed CCS calculus [31], in contrast to the refinement relation presented in this paper, which is not decidable. Interestingly, they also establish that time-abstracting *congruence* (i.e. equivalence in all contexts) is standard timed bisimulation. The corresponding result for this paper is that untimed traces congruence for timed processes is the same as (finite) timed failures equivalence.

Baeten and Bergstra [1] have considered the embedding of untimed ACP into real time ACP. They propose a translation of untimed ACP into the timed setting, for example translating a to $\int_{t \geqslant 0} a(t)$: an untimed a process specifies nothing about the time the a should occur, so it translates to the timed process that can perform an a at any time. This is also the philosophy of this paper. They also consider the translation of certain identities of ACP into the timed framework; this permits reasoning at a higher (untimed) level of abstraction to be incorporated when detailed reasoning about timing issues is also required.

6.2. Urgent events

Events in timed CSP are treated as non-urgent: processes are described in terms of when events become enabled and disabled. A process does not have complete *control* over the performance of (visible) events, since the cooperation of its environment is always required for an event to occur. The approach taken in timed CSP, and some other process algebras, is that the process therefore is not given *responsibility* for the performance of events; the most that can be expected of a process is that it is willing to go along with the occurrence of an event.

The other principal approach to events in timed process algebras, taken by Timed LOTOS [21, 2] among others, is to assume that the environment of a process will permit certain events to occur when the process wishes to perform them. Thus actions of a process may be treated as *urgent*, in that they occur as soon as they are enabled by the process. However, since the environmental assumption is not always justified – the process may be placed in parallel with a process which is unwilling to allow an urgent action to occur – it is possible that a catastrophic timelock occurs and no progress can be made. It is the responsibility of the system designer to ensure that no timelocks occur in a particular design. Alternatively, the parallel operator may treat urgent events as observable signals. In this treatment non-urgent events may synchronise with urgent ones but may not block them. This approach was taken in [7, 6].

In fact, urgent and non-urgent actions are incorporated within a single framework in [2], where actions are initially non-urgent, but may be made urgent by an *urgency* operator.

The question arises: how do the techniques developed here apply in the case of urgency?

The natural timed denotational semantics for systems with urgent actions is in terms of timed traces. Thus we might hope that the traces refinement discussed in this paper would remain applicable in the presence of urgent actions. Indeed, the may testing characterisation of the refinement relation yields the same results, and all of the timed LOTOS operators preserve the refinement relation. Also, the urgency operator $\rho(U)$ preserves refinement, in the sense that if P is refined by Q, then it is also refined by $\rho(U)(Q)$; making events urgent does not remove any safety properties. We might expect to use timewise refinement techniques when the requirements on a system split naturally into untimed safety properties, and timing requirements.

The notion of a timed failure is inapplicable to an urgent action, since it is performed as soon as it becomes possible. All the information about when a process was ready to perform it is contained in the timed trace; refusal information is redundant (though it remains appropriate for non-urgent actions). Thus, the failures refinement relation is not appropriate when dealing with urgent timed systems. This is reflected by the relation obtained by considering the relation $\underset{\sim_t}{\sqsubseteq}$ defined in terms of must testing. If process Q has an urgent action a to be performed at time t, then the test $T = Wait[(t+1)]; \omega \rightarrow Stop$ will yield a timelock at time t, since it will not cooperate with Q on the action a. However, any untimed process P (except an immediately divergent

one) has P must $\Theta(T)$. Thus no useful process P has $P \underset{\sim_f}{\sqsubset} Q$. The refinement relation collapses in the face of urgent actions, holding only when the untimed process is divergent.

In systems where both urgent and non-urgent events are present, it appears that the results for traces refinement apply as easily as in the pure non-urgent case of timed CSP. Failures refinement as discussed here will be useful only as far as the first urgent action. It is not even clear whether there is any more appropriate notion of failures style refinement, since timed liveness properties in urgent systems do not appear to have any untimed counterpart. The archetypical timed liveness property is that of timelock-freedom, and it is unlikely that this could be established by untimed analysis.

6.3. *Current and future work*

Earlier work [26] investigated the relationship between the untimed models and the standard timed failures model of [22]. The difficulties encountered in using that model to treat infinite behaviour led to the development of the infinite failures model, which supports a more natural treatment of timewise refinement from the untimed models.

Other refinement relations are also under investigation. In particular, a relation between the failures/divergences model and the timed failures stabilities model that treats instability as divergence has that all CSP operators preserve refinement; and this refinement relation is complete for stable processes. When stability considerations are important then this relation would be the natural one to use. Of particular interest is the relationship between the timed models and the timed probabilistic models for CSP developed by Lowe [16]. Work has already been done in this direction (see e.g. [17]), which it seems should fit into the framework presented in this paper.

The underlying theory presented here is of course more general than simply CSP, and should be applicable wherever processes are modelled in terms of the behaviours they may exhibit. It may for example be applicable to Gerth and Kuiper's interface refinement [11]. I feel that the theory will be useful only if refinement relations can be established at the syntactic level, since if refinement can be shown only by examining the semantics directly, then verifying abstract specifications of processes via refinement is unlikely to be much easier than performing the verification directly.

Acknowledgements

Thanks are due to Mike Mislove, Bill Roscoe, Mike Reed, Jim Davies, Dave Jackson, Michael Goldsmith, and members of the ESPRIT SPEC, REACT and CONCUR projects for useful discussions on this material. In particular, Bill Roscoe was responsible for the formulation of the alternating bit protocol presented here. Thanks are also due to the anonymous referees whose thorough reading and careful comments have been extremely useful.

This work was funded under SERC research fellowship B91/RFH/312. Additional funding was received from EC Basic Research Action 7166 CONCUR 2, and the US Office of Naval Research.

Appendix A. Semantic models and functions

A.1. Traces

The traces model $\mathcal{M}_{UT}$ is defined to be those sets of traces that are non-empty, and closed under prefixing. They are ordered under set containment.

A.2. The semantic function $\mathcal{F}_{UT}$

The space of environments is defined as

$$\mathcal{E}_{UT} = \mathcal{V} \to \mathcal{M}_{UT}$$

The semantic function

$$\mathcal{F}_{UT} : CSP \to \mathcal{E}_{UT} \to \mathcal{M}_{UT}$$

is defined by the following set of equations:

$$\begin{aligned}
\mathcal{F}_{UT}[\![Loop]\!]\rho &\mathrel{\widehat{=}} \{tr \in \Sigma^*\} \\
\mathcal{F}_{UT}[\![Stop]\!]\rho &\mathrel{\widehat{=}} \{\langle\rangle\} \\
\mathcal{F}_{UT}[\![Skip]\!]\rho &\mathrel{\widehat{=}} \{\langle\rangle, \langle\surd\rangle\} \\
\mathcal{F}_{UT}[\![P; Q]\!]\rho &\mathrel{\widehat{=}} \{tr \mid tr \in \mathcal{F}_{UT}[\![P]\!]\rho \wedge tr \upharpoonright \{\surd\} = \langle\rangle\} \\
&\quad \cup \\
&\quad \{tr_P \frown tr_Q \mid tr_P \frown \langle\surd\rangle \in \mathcal{F}_{UT}[\![P]\!]\rho \wedge \\
&\qquad tr_P \upharpoonright \{\surd\} = \langle\rangle \wedge tr_Q \in \mathcal{F}_{UT}[\![Q]\!]\rho\} \\
\mathcal{F}_{UT}[\![P \overset{t_0}{\rhd} Q]\!]\rho &\mathrel{\widehat{=}} \mathcal{F}_{UT}[\![P]\!]\rho \cup \mathcal{F}_{UT}[\![Q]\!]\rho \\
\mathcal{F}_{UT}[\![P \,\Box\, Q]\!]\rho &\mathrel{\widehat{=}} \mathcal{F}_{UT}[\![P]\!]\rho \cup \mathcal{F}_{UT}[\![Q]\!]\rho \\
\mathcal{F}_{UT}[\![a : A \to P_a]\!]\rho &\mathrel{\widehat{=}} \bigcup_{a \in A} \{\langle a \rangle \frown tr \mid tr \in \mathcal{F}_{UT}[\![P_a]\!]\rho\} \\
\mathcal{F}_{UT}\left[\!\!\left[\bigsqcap_{i \in I} P_i\right]\!\!\right]\rho &\mathrel{\widehat{=}} \bigcup_{i \in I} \mathcal{F}_{UT}[\![P_i]\!]\rho \\
\mathcal{F}_{UT}[\![P_A \|_B Q]\!]\rho &\mathrel{\widehat{=}} \{tr \in (A \cup B)^* \mid tr \upharpoonright A \in \mathcal{F}_{UT}[\![P]\!]\rho \wedge \\
&\qquad tr \upharpoonright B \in \mathcal{F}_{UT}[\![Q]\!]\rho\} \\
\mathcal{F}_{UT}[\![P \,|||\, Q]\!]\rho &\mathrel{\widehat{=}} \{tr \mid \exists tr_P \in \mathcal{F}_{UT}[\![P]\!]\rho, tr_Q \in \mathcal{F}_{UT}[\![Q]\!]\rho \bullet \\
&\qquad tr \ interleaves \ (tr_P, tr_Q)\}
\end{aligned}$$

$$\mathscr{F}_{UT}[\![P \setminus A]\!]\rho \mathrel{\widehat{=}} \{tr \setminus A \mid tr \in \mathscr{F}_{UT}[\![P]\!]\rho\}$$
$$\mathscr{F}_{UT}[\![f(P)]\!]\rho \mathrel{\widehat{=}} \{f(tr) \mid tr \in \mathscr{F}_{UT}[\![P]\!]\rho\}$$
$$\mathscr{F}_{UT}[\![f^{-1}(P)]\!]\rho \mathrel{\widehat{=}} \{tr \mid f(tr) \in \mathscr{F}_{UT}[\![P]\!]\rho\}$$
$$\mathscr{F}_{UT}[\![Y]\!]\rho \mathrel{\widehat{=}} \rho(Y)$$

where *interleaves* is defined on sequences $m, m_1, m_2 : \Sigma^* \cup \Sigma^\omega$ as follows:

$$\begin{aligned} m \; interleaves(m_1, m_2) \Leftrightarrow \#m = \#m_1 + \#m_2 \\ \wedge \exists f_1 : (1..\#m_1) \rightarrowtail_< (1..\#m), f_2 : (1..\#m_2) \rightarrowtail_< (1..\#m) \bullet \\ ran(f_1) \cap ran(f_2) = \{\} \\ \wedge ran(f_1) \cup ran(f_2) = 1..\#m \\ \wedge \forall n \in 1..\#m_1 \bullet m_1@n = m@f_1(n) \\ \wedge \forall n \in 1..\#m_2 \bullet m_2@n = m@f_2(n) \end{aligned}$$

where $1..n = \{j \mid 1 \leqslant j \leqslant n\}$ $1..\omega = \mathbf{N}$, $A \rightarrowtail_< B$ is the set of monotonically increasing injective total functions from A to B, $ran(f)$ is the range of function f and $m@n$ is the nth element of sequence m.

The semantics $\mathscr{F}_{UT}[\![\mu Y \circ P]\!]\rho$ of a recursive term is defined to be the least fixed point of the function

$$\lambda S \bullet \mathscr{F}_{UT}[\![P]\!](\rho[S/Y]) : \mathscr{M}_{UT} \rightarrow \mathscr{M}_{UT}$$

where substitution in an environment is defined by

$$\rho[S/Y](Z) = \rho(Z) \quad Z \neq Y$$
$$\rho[S/Y](Y) = S$$

A.3. Untimed infinite traces, failures and divergences

The process axioms given in [24] correspond to the following properties required of a set S for it to correspond to the set of observations of some process. Thus the semantic model $\mathscr{M}_{UI}$ is the collection of sets

$$S \subseteq \{f\} \times (\Sigma^* \times \mathbf{P}(\Sigma)) \cup \{d\} \times \Sigma^* \cup \{i\} \times \Sigma^\omega$$

(ordered under reverse containment) that meet these eight axioms:

(1) $(f,(tr_1 \frown tr_2, \{\})) \in S \Rightarrow (f,(tr_1, \{\})) \in S$
(2) $(f,(tr,X)) \in S \wedge Y \subseteq X \Rightarrow (f,(tr,Y)) \in S$
(3) $(f,(tr,X)) \in S \wedge \forall a \in Y \bullet (f,(tr \frown \langle a \rangle, \{\})) \notin S \Rightarrow (f,(tr, X \cup Y)) \in S$
(4) $(d,tr_1) \in S \Rightarrow (d,tr_1 \frown tr_2) \in S$
(5) $(d,tr_1) \in S \Rightarrow (f,(tr_1 \frown tr_2, X)) \in S$
(6) $(i,tr \frown u) \in S \Rightarrow (f,(tr,\{\})) \in S$
(7) $(d,tr) \in S \Rightarrow (i,tr \frown u) \in S$

$$(8)\ (f,(tr_1,\{\})) \in S \Rightarrow \exists T \bullet (\forall tr_2 \in T \bullet (f,(tr_1{}^\frown tr_2,\{a \mid tr_2{}^\frown\langle a\rangle \notin T\})) \subset S \wedge \{(i,tr_1{}^\frown u) \mid u \in \overline{T}\} \subseteq S)$$

Here $\overline{T} = \{u \in \Sigma^\omega \mid \forall tr < u \bullet tr \in T\}$, where T ranges over finite prefix closed sets of finite traces.

A.4. The semantic function $\mathscr{F}_{UI}$

The function $\mathscr{F}_{UI}$ is defined in terms of three functions $\mathscr{F}_{UD}$, $\mathscr{F}_{UF}$, and $\mathscr{F}_{I}$, yielding divergences, failures, and infinite traces respectively. It is then given by

$$\begin{aligned}\mathscr{F}_{UI}[\![P]\!]\rho = {} & \{(d,tr) \mid tr \in \mathscr{F}_{UD}[\![P]\!]\rho\} \\ & \cup \{(f,(tr,X)) \mid (tr,X) \in \mathscr{F}_{UF}[\![P]\!]\rho\} \\ & \cup \{(i,u) \mid u \in \mathscr{F}_{I}[\![P]\!]\rho\}\end{aligned}$$

A.5. The semantic function $\mathscr{F}_{UD}$

Define $\mathscr{E}_{UD} = \mathscr{V} \to \mathscr{M}_{UD}$. The semantic function

$$\mathscr{F}_{UD} : CSP \to \mathscr{E}_{UD} \to \mathscr{M}_{UD}$$

is defined by the following set of equations:

$$\mathscr{F}_{UD}[\![Loop]\!]\rho \mathrel{\widehat{=}} \{tr \mid tr \in \Sigma^*\}$$

$$\mathscr{F}_{UD}[\![Stop]\!]\rho \mathrel{\widehat{=}} \{\}$$

$$\mathscr{F}_{UD}[\![Skip]\!]\rho \mathrel{\widehat{=}} \{\}$$

$$\begin{aligned}\mathscr{F}_{UD}[\![P;Q]\!]\rho \mathrel{\widehat{=}} {} & \{tr{}^\frown tr' \mid tr \in \mathscr{F}_{UD}[\![P]\!]\rho \wedge \surd \notin \sigma(tr) \wedge tr' \in \Sigma^*\} \\ & \cup \\ & \{tr{}^\frown tr' \mid (tr{}^\frown\langle\surd\rangle,\{\}) \in \mathscr{F}_{UF}[\![P]\!]\rho \wedge \surd \notin \sigma(tr) \\ & \qquad \wedge tr' \in \mathscr{F}_{UD}[\![Q]\!]\rho\}\end{aligned}$$

$$\mathscr{F}_{UD}[\![P \overset{t_0}{\rhd} Q]\!]\rho \mathrel{\widehat{=}} \mathscr{F}_{UD}[\![P]\!]\rho \cup \mathscr{F}_{UD}[\![Q]\!]\rho$$

$$\mathscr{F}_{UD}[\![P \,\Box\, Q]\!]\rho \mathrel{\widehat{=}} \mathscr{F}_{UD}[\![P]\!]\rho \cup \mathscr{F}_{UD}[\![Q]\!]\rho$$

$$\mathscr{F}_{UD}[\![a : A \to P_a]\!]\rho \mathrel{\widehat{=}} \bigcup_{a \in A} \{\langle a\rangle{}^\frown tr \mid tr \in \mathscr{F}_{UD}[\![P_a]\!]\rho\}$$

$$\mathscr{F}_{UD}\left[\!\!\left[\prod_{i \in I} P_i\right]\!\!\right]\rho \mathrel{\widehat{=}} \bigcup_{i \in I} \mathscr{F}_{UD}[\![P_i]\!]\rho$$

$$\begin{aligned}\mathscr{F}_{UD}[\![P_A \|_B Q]\!]\rho \mathrel{\widehat{=}} \{ & tr{}^\frown tr' \mid tr \in (A \cup B)^* \\ & \wedge \\ & (tr \upharpoonright A,\{\}) \in \mathscr{F}_{UF}[\![P]\!]\rho \wedge tr \upharpoonright B \in \mathscr{F}_{UD}[\![Q]\!]\rho \\ & \vee tr \upharpoonright A \in \mathscr{F}_{UD}[\![P]\!]\rho \wedge (tr \upharpoonright B,\{\}) \in \mathscr{F}_{UF}[\![Q]\!]\rho\}\end{aligned}$$

$$\mathscr{F}_{UD}[\![P \parallel Q]\!]\rho \mathrel{\hat{=}} \{tr \mid \exists tr_P \in \mathscr{F}_{UD}[\![P]\!]\rho, (tr_Q, \{\}) \in \mathscr{F}_{UF}[\![Q]\!]\rho \bullet tr \text{ interleaves } (tr_P, tr_Q)\}$$
$$\cup$$
$$\{tr \mid \exists (tr_P, \{\}) \in \mathscr{F}_{UF}[\![P]\!]\rho, tr_Q \in \mathscr{F}_{UD}[\![Q]\!]\rho \bullet tr \text{ interleaves } (tr_P, tr_Q)\}$$

$$\mathscr{F}_{UD}[\![P \setminus A]\!]\rho \mathrel{\hat{=}} \{tr \setminus A^\frown tr' \mid tr \in \mathscr{F}_{UD}[\![P]\!]\rho\}$$
$$\cup$$
$$\{u \setminus A^\frown tr' \mid u \in \mathscr{F}_I[\![P]\!]\rho \wedge \#(u \setminus A) < \infty\}$$

$$\mathscr{F}_{UD}[\![f(P)]\!]\rho \mathrel{\hat{=}} \{f(tr)^\frown tr' \mid tr \in \mathscr{F}_{UD}[\![P]\!]\rho\}$$

$$\mathscr{F}_{UD}[\![f^{-1}(P)]\!]\rho \mathrel{\hat{=}} \{tr \mid f(tr) \in \mathscr{F}_{UD}[\![P]\!]\rho\}$$

$$\mathscr{F}_{UD}[\![Y]\!]\rho \mathrel{\hat{=}} \rho(Y)$$

A.6. The semantic function $\mathscr{F}_I$

Define $\mathscr{E}_I = \mathscr{V} \to \mathscr{M}_I$. The semantic function

$$\mathscr{F}_I : CSP \to \mathscr{E}_I \to \mathscr{M}_I$$

is defined by the following set of equations:

$$\mathscr{F}_I[\![Loop]\!]\rho \mathrel{\hat{=}} \{u \mid u \in \Sigma^* \cup \Sigma^\omega\}$$
$$\mathscr{F}_I[\![Stop]\!]\rho \mathrel{\hat{=}} \{\}$$
$$\mathscr{F}_I[\![Skip]\!]\rho \mathrel{\hat{=}} \{\}$$
$$\mathscr{F}_I[\![P; Q]\!]\rho \mathrel{\hat{=}} \{u \mid u \in \mathscr{F}_I[\![P]\!]\rho \wedge \surd \notin \sigma(u)\}$$
$$\cup$$
$$\{tr^\frown u' \mid (tr^\frown\langle\surd\rangle, \{\}) \in \mathscr{F}_{UF}[\![P]\!]\rho \wedge \surd \notin \sigma(tr) \wedge u' \in \mathscr{F}_I[\![Q]\!]\rho\}$$
$$\cup$$
$$\{tr^\frown u \mid tr \in \mathscr{F}_{UD}[\![P; Q]\!]\rho\}$$

$$\mathscr{F}_I[\![P \overset{t_0}{\triangleright} Q]\!]\rho \mathrel{\hat{=}} \mathscr{F}_I[\![P]\!]\rho \cup \mathscr{F}_I[\![Q]\!]\rho$$

$$\mathscr{F}_I[\![P \,\Box\, Q]\!]\rho \mathrel{\hat{=}} \mathscr{F}_I[\![P]\!]\rho \cup \mathscr{F}_I[\![Q]\!]\rho$$

$$\mathscr{F}_I[\![a : A \to P_a]\!]\rho \mathrel{\hat{=}} \bigcup_{a \in A} \{\langle a\rangle^\frown u \mid u \in \mathscr{F}_I[\![P_a]\!]\rho\}$$

$$\mathscr{F}_I\Big[\!\Big[\bigsqcap_{i \in I} P_i\Big]\!\Big]\rho \mathrel{\hat{=}} \bigcup_{i \in I} \mathscr{F}_I[\![P_i]\!]\rho$$

$$\begin{aligned}
\mathscr{F}_I[\![P_A \|_B Q]\!]\rho \mathrel{\hat{=}} \{u \mid\ & u \in (A \cup B)^{\omega} \\
& \wedge u \upharpoonright A \in \mathscr{F}_I[\![P]\!]\rho \vee (u \upharpoonright A, \{\}) \in \mathscr{F}_{UF}[\![P]\!]\rho \\
& \wedge u \upharpoonright B \in \mathscr{F}_I[\![Q]\!]\rho \vee (u \upharpoonright B, \{\}) \in \mathscr{F}_{UF}[\![Q]\!]\rho \\
& \} \\
\cup\ & \\
& \{tr^\frown u \mid tr \in \mathscr{F}_{UD}[\![P_A \|_B Q]\!]\rho\} \\
\mathscr{F}_I[\![P \,|||\, Q]\!]\rho \mathrel{\hat{=}} \{u \mid\ & \exists u_P, u_Q \bullet (\#u_P = \infty \vee \#u_Q = \infty) \\
& \wedge\ u\ interleaves(u_P, u_Q) \\
& \wedge (u_P, \{\}) \in \mathscr{F}_{UF}[\![P]\!]\rho \vee u_P \in \mathscr{F}_I[\![P]\!]\rho \\
& \wedge (u_Q, \{\}) \in \mathscr{F}_{UF}[\![Q]\!]\rho \vee u_Q \in \mathscr{F}_I[\![Q]\!]\rho\} \\
\cup\ & \\
& \{tr^\frown u \mid tr \in \mathscr{F}_{UD}[\![P \,|||\, Q]\!]\rho\} \\
\mathscr{F}_I[\![P \setminus A]\!]\rho \mathrel{\hat{=}} & \{u \setminus A \mid u \in \mathscr{F}_I[\![P]\!]\rho \wedge \#(u \setminus A) = \infty\} \\
\cup\ & \\
& \{tr^\frown u \mid tr \in \mathscr{F}_{UD}[\![P \setminus A]\!]\rho\} \\
\mathscr{F}_I[\![f(P)]\!]\rho \mathrel{\hat{=}} & \{f(u)^\frown \mid u \in \mathscr{F}_I[\![P]\!]\rho\} \\
\cup\ & \\
& \{tr^\frown u \mid tr \in \mathscr{F}_{UD}[\![f(P)]\!]\rho\} \\
\mathscr{F}_I[\![f^{-1}(P)]\!]\rho \mathrel{\hat{=}} & \{u \mid f(u) \in \mathscr{F}_I[\![P]\!]\rho\} \\
\mathscr{F}_I[\![Y]\!]\rho \mathrel{\hat{=}} & \rho(Y)
\end{aligned}$$

A.7. The semantic function $\mathscr{F}_{UF}$

Define $\mathscr{E}_{UF} = \mathscr{V} \to \mathscr{M}_{UF}$. The semantic function

$$\mathscr{F}_{UF} : CSP \to \mathscr{E}_{UF} \to \mathscr{M}_{UF}$$

is defined by the following set of equations:

$$\begin{aligned}
\mathscr{F}_{UF}[\![Loop]\!]\rho &\mathrel{\hat{=}} \{(tr, X) \mid tr \in \Sigma^* \wedge X \subseteq \Sigma\} \\
\mathscr{F}_{UF}[\![Stop]\!]\rho &\mathrel{\hat{=}} \{(\langle\rangle, X) \mid X \subseteq \Sigma\} \\
\mathscr{F}_{UF}[\![Skip]\!]\rho &\mathrel{\hat{=}} \{(\langle\rangle, X) \mid \surd \notin X\} \\
&\cup \\
&\{(\langle\surd\rangle, X) \mid X \subseteq \Sigma\} \\
\mathscr{F}_{UF}[\![P; Q]\!]\rho &\mathrel{\hat{=}} \{(tr, X) \mid \surd \notin \sigma(tr) \wedge \\
&\qquad\qquad (tr, X \cup \{\surd\}) \in \mathscr{F}_{UF}[\![P]\!]\rho\} \\
&\cup
\end{aligned}$$

$$\{(tr^\frown tr', X) \mid \surd \notin \sigma(tr) \wedge (tr^\frown\langle\surd\rangle, \{\}) \in \mathscr{F}_{UF}[\![P]\!]\rho \wedge (tr', X) \in \mathscr{F}_{UF}[\![Q]\!]\rho\}$$

$$\cup$$

$$\{(tr, X) \mid tr \in \mathscr{F}_{UD}[\![P; Q]\!]\rho\}$$

$$\mathscr{F}_{UF}[\![P \overset{t_0}{\triangleright} Q]\!]\rho \mathrel{\hat{=}} \mathscr{F}_{UF}[\![Q]\!]\rho \cup \{(tr, X) \mid (tr, X) \in \mathscr{F}_{UF}[\![P]\!]\rho \wedge tr \neq \langle\rangle\}$$

$$\mathscr{F}_{UF}[\![P \,\Box\, Q]\!]\rho \mathrel{\hat{=}} \{(\langle\rangle, X) \mid (\langle\rangle, X) \in \mathscr{F}_{UF}[\![P]\!]\rho \cap \mathscr{F}_{UF}[\![Q]\!]\rho\}$$

$$\cup$$

$$\{(tr, X) \mid tr \neq \langle\rangle \wedge (tr, X) \in \mathscr{F}_{UF}[\![P]\!]\rho \cup \mathscr{F}_{UF}[\![Q]\!]\rho\}$$

$$\mathscr{F}_{UF}[\![a : A \rightarrow P_a]\!]\rho \mathrel{\hat{=}} \{(\langle\rangle, X) \mid X \cap A = \{\}\}$$

$$\cup$$

$$\bigcup_{a \in A} \{(\langle a\rangle^\frown tr, X) \mid (tr, X) \in \mathscr{F}_{UF}[\![P_a]\!]\rho\}$$

$$\mathscr{F}_{UF}\left[\!\!\left[\bigsqcap_{i \in I} P_i\right]\!\!\right]\rho \mathrel{\hat{=}} \bigcup_{i \in I} \mathscr{F}_{UF}[\![P_i]\!]\rho$$

$$\mathscr{F}_{UF}[\![P_A \parallel_B Q]\!]\rho \mathrel{\hat{=}} \{(tr, Z) \mid \exists X, Y \bullet (tr \restriction A, X \restriction A) \in \mathscr{F}_{UF}[\![P]\!]\rho \wedge (tr \restriction B, Y \restriction B) \in \mathscr{F}_{UF}[\![Q]\!]\rho \wedge (X \restriction A) \cup (Y \restriction B) = Z \restriction A \cup B \wedge tr = tr \restriction (A \cup B)\}$$

$$\cup$$

$$\{(tr, X) \mid tr \in \mathscr{F}_{UD}[\![P_A \parallel_B Q]\!]\rho\}$$

$$\mathscr{F}_{UF}[\![P \,|||\, Q]\!]\rho \mathrel{\hat{=}} \{(tr, X) \mid \exists tr_P, tr_Q \bullet tr \ interleaves(tr_P, tr_Q) \wedge (tr_P, X) \in \mathscr{F}_{UF}[\![P]\!]\rho \wedge (tr_Q, X) \in \mathscr{F}_{UF}[\![Q]\!]\rho\}$$

$$\cup$$

$$\{(tr, X) \mid tr \in \mathscr{F}_{UD}[\![P \,|||\, Q]\!]\rho\}$$

$$\mathscr{F}_{UF}[\![P \setminus A]\!]\rho \mathrel{\hat{=}} \{(tr \setminus A, X) \mid (tr, X \cup A) \in \mathscr{F}_{UF}[\![P]\!]\rho\}$$

$$\cup$$

$$\{(tr, X) \mid tr \in \mathscr{F}_{UD}[\![P \setminus A]\!]\rho\}$$

$$\mathscr{F}_{UF}[\![f(P)]\!]\rho \mathrel{\hat{=}} \{(f(tr), X) \mid (tr, f^{-1}(X)) \in \mathscr{F}_{UF}[\![P]\!]\rho\}$$

$$\cup$$

$$\{(tr, X) \mid tr \in \mathscr{F}_{UD}[f(P)]\!]\rho\}$$

$$\mathscr{F}_{UF}[\![f^{-1}(P)]\!]\rho \mathrel{\hat{=}} \{(tr, X) \mid (f(tr), f(X)) \in \mathscr{F}_{UF}[\![P]\!]\rho\}$$

$$\mathscr{F}_{UF}[\![Y]\!]\rho \mathrel{\hat{=}} \rho(Y)$$

The semantics $\mathscr{F}_{UI}[\![\mu Y \circ P]\!]\rho$ of a recursive term is defined to be the least fixed point of the function

$$F = \lambda S \bullet \mathscr{F}_I[\![P]\!](\rho[S/Y]) : \mathscr{M}_{UI} \to \mathscr{M}_{UI}$$

It is established in [24] that this is well-defined and that this is equal to $F^{\alpha}(\perp)$ for some ordinal α, where $\perp = \mathscr{F}_{UI}[\![Loop]\!]\rho$.

A.8. *Infinite timed failures*

The information ordering on behaviours is defined as follows:

$$(s', \aleph') \preccurlyeq (s, \aleph) \Leftrightarrow \exists s'' \bullet s = s' \frown s'' \wedge \aleph' \subseteq \aleph \lhd \mathit{begin}(s'')$$

We formally define $\mathscr{M}_{TI}$ to be those subsets S of $T\Sigma^{\omega}_{\leqslant} \times \mathit{IRSET}$ satisfying axioms 1–3 given below, and axiom 4 to follow.

1. $(\langle\rangle, \{\}) \in S$
2. $(s, \aleph) \in S \wedge (s', \aleph') \preccurlyeq (s, \aleph) \Rightarrow (s', \aleph') \in S$
3. $(s, \aleph) \in S \Rightarrow$
$$\exists \aleph' \in \mathit{IRSET} \bullet \aleph \subseteq \aleph' \wedge (s, \aleph') \in S \wedge \forall (t, a) \in \mathbf{R}_+ \times \Sigma \bullet$$
$$(t, a) \notin \aleph' \Rightarrow (s \lhd t \frown \langle (t, a) \rangle, \aleph' \lhd t) \in S$$
$$\wedge$$
$$(t > 0 \wedge \neg \exists \varepsilon > 0 \bullet ((t - \varepsilon, t) \times \{a\} \subseteq \aleph'))$$
$$\Rightarrow (s \lhd t \frown \langle (t, a) \rangle, \aleph' \lhd t) \in S$$

Axioms 1 and 2 require that an element of $\mathscr{M}_{TI}$ must be a non-empty downward closed set of behaviours. Axiom 3 requires that on every execution, timed events must be either possible or refusable.

A set of behaviours T is finitely variable if for every time t, the set $T \lhd t$ is a complete partial order under $\preccurlyeq$. A set of behaviours T is closed if

$$T = \overline{T} = \{(s, \aleph) \mid \forall t \bullet (s, \aleph) \lhd t \in T\}$$

Let $\mathscr{CL}$ be the set of finitely variable closed sets of behaviours satisfying axioms 1–3. Then axiom 4 states that

4. $S = \bigcup \{Q \in \mathscr{CL} \mid S \supseteq Q\}$

A.9. *The semantic function* $\mathscr{F}_{TI}$

Define $\mathscr{E}_{TI} = \mathscr{V} \to \mathscr{M}_{TI}$. The semantic function

$$\mathscr{F}_{TI} : \mathit{CSP} \to \mathscr{E}_{TI} \to \mathscr{M}_{TI}$$

is defined by the following set of equations:

$$\mathscr{F}_{TI}[\![Loop]\!]\rho \mathrel{\widehat{=}} \{(s,\aleph) \mid s \in T\Sigma^{\omega}_{\leqslant} \wedge \aleph \in IRSET\}$$

$$\mathscr{F}_{TI}[\![Stop]\!]\rho \mathrel{\widehat{=}} \{(\langle\rangle,\aleph) \mid \aleph \in IRSET\}$$

$$\mathscr{F}_{TI}[\![Skip]\!]\rho \mathrel{\widehat{=}} \{(\langle\rangle,\aleph) \mid \surd \notin \sigma(\aleph)\} \cup \{(\langle(t,\surd)\rangle,\aleph) \mid t \geqslant 0 \wedge \surd \notin \sigma(\aleph \uparrow [0,t))\}$$

$$\begin{aligned}\mathscr{F}_{TI}[\![P;Q]\!]\rho \mathrel{\widehat{=}} \{(s,\aleph) \mid\ & \surd \notin \sigma(s) \wedge \\ & (s, \aleph \cup ([0, end(s,\aleph)) \times \{\surd\})) \in \mathscr{F}_{TI}[\![P]\!]\rho \\ & \vee \\ & s = s_P \frown s_Q \wedge \surd \notin \sigma(s_P) \wedge \\ & (s_Q, \aleph) - t \in \mathscr{F}_{TI}[\![Q]\!]\rho \wedge \\ & (s_P \frown \langle(t,\surd)\rangle, \aleph \lhd t \cup ([0,t) \times \{\surd\})) \in \mathscr{F}_{TI}[\![P]\!]\rho\}\end{aligned}$$

$$\begin{aligned}\mathscr{F}_{TI}[\![P \overset{t_0}{\rhd} Q]\!]\rho \mathrel{\widehat{=}} & \{(s,\aleph) \mid begin(s) \leqslant t_0 \wedge (s,\aleph) \in \mathscr{F}_{TI}[\![P]\!]\rho\} \\ \cup\ & \{(s,\aleph) \mid begin(s) \geqslant t_0 \wedge (\langle\rangle, \aleph \lhd t_0) \in \mathscr{F}_{TI}[\![P]\!]\rho \\ & \wedge \\ & (s,\aleph) - t_0 \in \mathscr{F}_{TI}[\![Q]\!]\rho\}\end{aligned}$$

$$\begin{aligned}\mathscr{F}_{TI}[\![P \,\Box\, Q]\!]\rho \mathrel{\widehat{=}} & \{(\langle\rangle,\aleph) \mid (\langle\rangle,\aleph) \in \mathscr{F}_{TI}[\![P]\!]\rho \cap \mathscr{F}_{TI}[\![Q]\!]\rho\} \\ \cup\ & \{(s,\aleph) \mid s \neq \langle\rangle \wedge (s,\aleph) \in \mathscr{F}_{TI}[\![P]\!]\rho \cup \mathscr{F}_{TI}[\![Q]\!]\rho \\ & \wedge \\ & (\langle\rangle, \aleph \lhd begin(s)) \in \mathscr{F}_{TI}[\![P]\!]\rho \cap \mathscr{F}_{TI}[\![Q]\!]\rho\}\end{aligned}$$

$$\begin{aligned}\mathscr{F}_{TI}[\![a : A \to P_a]\!]\rho = & \{(\langle\rangle,\aleph) \mid A \cap \sigma(\aleph) = \{\}\} \\ \cup\ & \{(\langle(t,a)\rangle \frown (s+t), \aleph) \mid \\ & \qquad a \in A \wedge t \geqslant 0 \wedge A \cap \sigma(\aleph \lhd t) = \{\} \\ & \qquad \wedge (s, \aleph - t) \in \mathscr{F}_{TI}[\![P_a]\!]\rho\}\end{aligned}$$

$$\mathscr{F}_{TI}\Big[\!\Big[\bigsqcap_{i\in I} P_i\Big]\!\Big]\rho \mathrel{\widehat{=}} \bigcup_{i\in I} \mathscr{F}_{TI}[\![P_i]\!]\rho$$

$$\begin{aligned}\mathscr{F}_{TI}[\![P_A \parallel_B Q]\!]\rho \mathrel{\widehat{=}} \{(s,\aleph) \mid\ & \exists \aleph_P, \aleph_Q \bullet \\ & \aleph \upharpoonright (A \cup B) = (\aleph_P \upharpoonright A) \cup (\aleph_Q \upharpoonright B) \\ & \wedge s = s \upharpoonright (A \cup B) \\ & \wedge (s \upharpoonright A, \aleph_P) \in \mathscr{F}_{TI}[\![P]\!]\rho \\ & \wedge (s \upharpoonright B, \aleph_Q) \in \mathscr{F}_{TI}[\![Q]\!]\rho\ \}\end{aligned}$$

$$\mathscr{F}_{TI}[\![P \parallel Q]\!]\rho \mathrel{\widehat{=}} \{(s,\aleph) \mid \exists s_P, s_Q \bullet \forall t \bullet strip(s \uparrow t) interleaves(strip(s_P \uparrow t), strip(s_Q \uparrow t)) \wedge (s_P,\aleph) \in \mathscr{F}_{TI}[\![P]\!]\rho \wedge (s_Q,\aleph) \in \mathscr{F}_{TI}[\![Q]\!]\rho\}$$

$$\mathscr{F}_{TI}[\![P \setminus A]\!]\rho \mathrel{\widehat{=}} \{(s \setminus A,\aleph) \mid (s,\aleph \cup ([0,\infty) \times A) \in \mathscr{F}_{TI}[\![P]\!]\rho\}$$

$$\mathscr{F}_{TI}[\![f(P)]\!]\rho \mathrel{\widehat{=}} \{(f(s),\aleph) \mid (s, f^{-1}(\aleph)) \in \mathscr{F}_{TI}[\![P]\!]\rho\}$$

$$\mathscr{F}_{TI}[\![f^{-1}(P)]\!]\rho \mathrel{\widehat{=}} \{(s,\aleph) \mid (f(s), f(\aleph)) \in \mathscr{F}_{TI}[\![P]\!]\rho\}$$

$$\mathscr{F}_{TI}[\![Y]\!]\rho \mathrel{\widehat{=}} \rho(Y)$$

The semantics $\mathscr{F}_{TI}[\![\mu Y \circ P]\!]\rho$ for a recursive term is defined to be the least fixed point of the function

$$\lambda S \bullet \mathscr{F}_{TI}[\![P]\!](\rho[S/Y]) : \mathscr{M}_{TI} \to \mathscr{M}_{TI}$$

It is established in [20] that this is well-defined.

References

[1] J.C.M. Baeten and J.A. Bergstra, Discrete time process algebra, University of Amsterdam, Report P9208b, 1992.

[2] T. Bolognesi and F. Lucidi, Timed process algebras with urgent interactions and a unique powerful binary operator, Lecture Notes in Computer Science, Vol. 600 (Springer, Berlin, 1992).

[3] S.D. Brookes and A.W. Roscoe, An improved failures model for communicating sequential processes, *Proc. Seminar on Concurrency*, Lecture Notes in Computer Science, Vol. 197 (Springer, Berlin, 1985).

[4] S.D. Brookes, A.W. Roscoe and D.J. Walker, An operational semantics for CSP, submitted for publication, 1992.

[5] R. Cleaveland and A.E. Zwarico, *A Theory of Testing for Real-time* (North Carolina SU and Johns Hopkins, 1992).

[6] J.W. Davies, *Specification and Proof in Real-time CSP* (Cambridge Univ. Press Cambidge, 1993).

[7] J.W. Davies, D.M. Jackson and S.A. Schneider, Making things happen in Timed CSP, PRG Tech. Report 2-90, Oxford, 1990.

[8] J.W. Davies and S.A. Schneider, Recursion induction for real-time processes, *Formal Aspects of Comput.* **5** (6) (1993).

[9] J.W. Davies and S.A. Schneider, A brief history of timed CSP, *Theoret. Comput. Sci.* **138** (1995).

[10] Formal Systems (Europe) Ltd., The Failures Divergences Refinement User Manual, 1993.

[11] R. Gerth, R. Kuiper and J. Segers, Interface refinement in reactive systems, *Proc. CONCUR '92*, Lecture Notes in Computer Science, Vol. 630 (Springer, Berlin, 1992).

[12] M. Hennessy, *Algebraic Theory of Processes* (MIT Press, Cambridge, 1988).

[13] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice-Hall, Englewood Cliffs, NJ, 1985).

[14] K.G. Larsen and Wang Yi, Time abstracted bisimulation: implicit specifications and decidability, *Proc. MFPS '93*, Lecture Notes in Computer Science Vol. 802 (Springer, Berlin, 1993).

[15] N.G. Leveson and J.L. Stolzy, Safety analysis using Petri nets, *IEEE Trans. Software Engineering*, (1987).

[16] G. Lowe, Prioritized and probabilistic models of timed CSP, Oxford University Computing Laboratory Tech. Report PRG-TR-24-91, 1991.

[17] G. Lowe, *Relating the Prioritized Model of Timed CSP to the Timed Failures Model* (Oxford University Computing Laboratory, Oxford, 1992).

[18] N. Lynch and F. Vaandrager, Forward and backward simulations for timing based systems, in: *Proc. Real-time: Theory in Practise*, Lecture Notes in Computer Science, Vol. 600 (Springer, Berlin, 1992).

[19] R. Milner, *Communication and Concurrency* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[20] M.W. Mislove, A.W. Roscoe and S.A. Schneider, Fixed points without completeness, *Theoret. Comput. Sci.* **138** (1995).
[21] J. Quemada and A. Fernandez, Introduction of quantitative relative time into LOTOS, in: *Protocol Specification, Testing and Verification VII* (North-Holland, Amsterdam, 1987).
[22] G.M. Reed, A uniform mathematical theory for real-time distributed computing, Oxford University DPhil Thesis, 1988.
[23] G.M. Reed and A.W. Roscoe, A timed model for communicating sequential processes, *Proc. 13th ICALP*, Lecture Notes in Computer Science, Vol. 226 (Springer, Berlin, 1986).
[24] A.W. Roscoe, Unbounded Nondeterminism in CSP, Oxford University Computing Laboratory. Tech. monograph PRG-67, 1988.
[25] A.W. Roscoe, Model-checking CSP, in: A.W. Roscoe, ed., *A Classical Mind* (Prentice-Hall, Englewood Cliffs, NJ, 1994).
[26] S.A. Schneider, correctness and communication in real-time systems, Oxford University DPhil. Thesis 1989.
[27] S.A. Schneider, An operational semantics for timed CSP, *Proc. Workshop on Concurrency*, Report 63, Programming Methodology Group, Chalmers University, 1992.
[28] S.A. Schneider, Unbounded nondeterminism for real-time processes, Oxford University Tech. Report 13–92, 1992.
[29] S.A. Schneider, An operational semantics for timed CSP, *Inform. Comput.* **116** (1995).
[30] S.A. Schneider, Specification and verification in Timed CSP, in: *Real-time Systems: Specification, Verification and Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1996).
[31] Wang Yi, Real-time behaviour of asynchronous agents, *Proc. CONCUR '90*, Lecture Notes in Computer Science, Vol. 458 (Springer, Berlin, 1990).